

# **Simulation der Deformation und Bewegung von Kleidung**

## **Studienarbeit**

vorgelegt von  
Christiane Lantermann



Institut für Computervisualistik  
Arbeitsgruppe Computergrafik

Betreuer und Prüfer: Prof. Dr.-Ing. Stefan Müller

Dezember 2003



## Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>5</b>
<b>2. Simulationsmodelle</b>	<b>6</b>
2.1. Einführung	6
2.1.1. Kurzer Einblick in die Dynamik	6
2.1.2. Aufbau und Komponenten eines mechanischen Simulationssystems	6
2.2. Mechanische Parameter des Stoffes	8
2.2.1. Gewebestruktur	8
2.2.2. Grundlegende mechanische Eigenschaften	9
2.3. Implementierung eines mechanischen Modells	11
2.3.1. Definition der Verhaltensgesetze	11
2.3.2. Fundamentale Gesetze der Mechanik	13
2.3.3. Definition eines Simulationsschemas	14
2.4. Mechanische Simulationssysteme	17
2.4.1. Ein gutes Simulationssystem	17
2.4.2. Geometrische Modelle	21
2.4.3. Kontinuum-Mechanik-Modelle	22
2.4.4. Partikelsystem-Modelle	26
2.4.5. Ein schnelles Partikel-System für irreguläre Meshes	30
2.5. Numerische Integration	40
2.5.1. Integrationstechniken	41
2.5.2. Auswahl der geeigneten Integrationsmethode	52
<b>3. Realisierung eines Partikelsystems</b>	<b>54</b>
3.1. Softwaretechnischer Entwurf	54
3.1.1. Aufbau des Partikelsystems mit seinen Klassen	54
3.1.2. Wichtige Methoden	60
3.2. UML-Diagramm des Partikel-Systems	63
3.3. Simulationsalgorithmus	64
3.3.1. Kräfte	64
3.3.2. Richtige Wahl der verschiedenen Konstanten	68
3.3.3. Integration: Eulermethode erster Ordnung	69

<b>4. Ergebnisse</b>	<b>71</b>
4.1. fallendes Tuch	71
4.2. Tischdecke	71
4.3. Fahne	72
<b>5. Fazit und Ausblick</b>	<b>73</b>
<b>6. Abbildungen</b>	<b>74</b>
<b>7. Literaturverzeichnis</b>	<b>91</b>

## **1. Einleitung**

In der heutigen Zeit werden immer mehr virtuelle Welten erstellt. Nicht nur Kinofilme, auch Werbung oder animierte Sicherheitsanweisungen in Flugzeugen werden durch virtuelle Charaktere dargestellt, die sich wiederum in einer virtuellen Welt befinden. Eine große Rolle spielt dabei natürlich die Kleidung der Akteure. Solange sie eng anliegende Kleidung tragen, müssen sich Hose und Oberteil nur dem Körper anpassen. Röcke oder Kleider hingegen bewegen sich schon bei leichten Windstößen. Wenn Textilien und Kleidung realitätsnah dargestellt werden sollen, müssen sie in ein Simulationsverfahren integriert werden.

Im Rahmen dieser Arbeit werden verschiedene Methoden zur Simulation von Bekleidung untersucht. Ziel ist es, auf Basis einer Analyse verschiedener Verfahren eine geeignete Methode auszuwählen, zu implementieren und die Ergebnisse an ausgewählten Beispielen mit möglichst ansprechender Qualität zu demonstrieren.

## **2. Simulationsmodelle**

Ein mechanisches Modell animiert die Stoffoberfläche so, dass sie sich wie ein reales Gewebe verhält.

In diesem Kapitel werden zunächst die grundlegenden Konzepte der mechanischen Simulation eingeführt. Daraufhin werden verschiedene Möglichkeiten beschrieben, Stoffmaterialien zu simulieren.

### **2.1. Einführung**

Von allen Bereichen der Physik ist nur die Mechanik (Newton) notwendig für die Simulation von Stoffen.

#### **2.1.1. Kurzer Einblick in die Dynamik**

Dynamik nennt man den Teil der Mechanik, der unter Einbeziehung der Parameter Zeit und Kraft die Entfaltung und Bewegung von Objekten untersucht. In der Kinematik sind die drei wichtigsten Größen die Position, Geschwindigkeit und Beschleunigung. Die Dynamik benötigt Kraft, Energie und Impuls, da sie die Masse berücksichtigt. Das fundamentale Gesetz der Dynamik ist die Gleichheit von Beschleunigung und dem Quotienten aus Kraft und Masse.

#### **2.1.2. Aufbau und Komponenten eines mechanischen Simulationssystems**

##### 1. Die Simulationsschleife

Ziel der mechanischen Simulation ist die virtuelle Repräsentation des mechanischen Verhaltens eines Stoff-Objekts, das mit seiner Umgebung interagiert. Zu einem gegebenen Zeitpunkt ist der aktuelle Zustand des Objekts durch seine Position und Geschwindigkeit definiert. Ausgehend von diesen Parametern wird der aktuelle Deformationszustand berechnet.

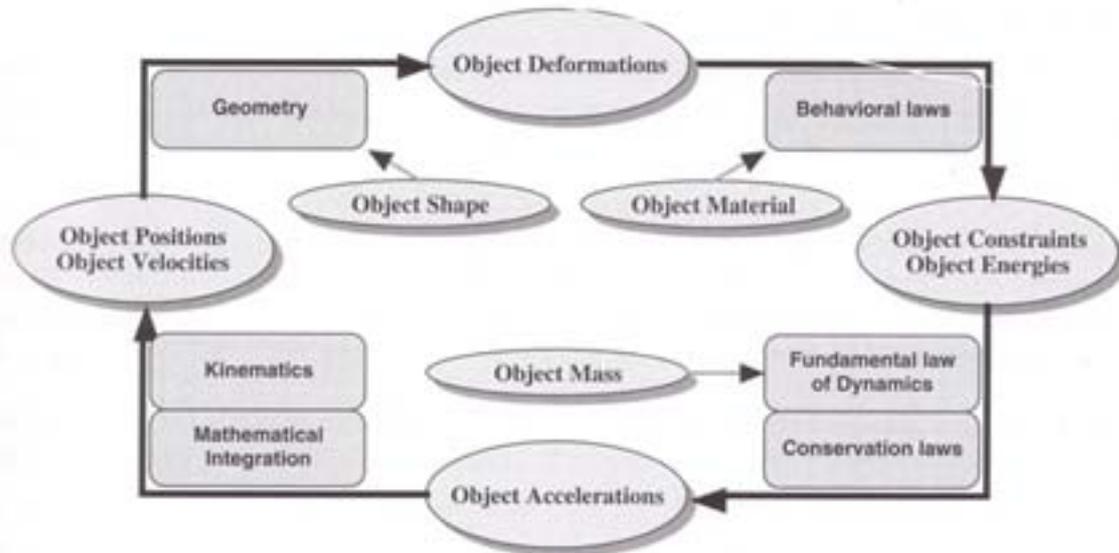


Abb.1: Die Komponenten eines mechanischen Systems

## 2. Ein mechanisches Simulationssystem und seine Komponenten

Für den Aufbau eines Stoff-Simulationssystems gibt es mehrere Ansätze.

Die materielle Analyse untersucht die Haupteigenschaften der Mechanik des Gewebematerials und seine Reaktionen auf gegebene Deformationen.

Das mechanische Modellieren nähert sich mithilfe verschiedener Verhaltensgesetze an das mechanische Verhalten an. Diese Gesetze können anhand analytischer und mathematischer Relationen – wie die Einschränkungen und Deformationen interner Kräfte im Material – mit einer gegebenen Menge von Parametern ausgedrückt werden. Dabei spezifizieren diese Parameter das Charakteristische des zu simulierenden Materials.

Das geometrische Modellieren definiert eine angemessene, geometrische Darstellung des Stoffes, die dafür entworfen wird, die Form sehr genau darzustellen. Das Simulationsprogramm legt einen effizienten geometrischen und mathematischen Rahmen fest, der das mechanische Modell mit den Gesetzen der Dynamik verbindet. Schließlich wird eine effiziente, numerische Integration zur Berechnung der Entwicklung des Systems implementiert. Dazu werden mechanische Gesetze verwendet, die das System beschreiben.

All diese Ansätze spielen bei der Qualität eines guten mechanischen Systems eine Rolle, die einen Kompromiss aus Genauigkeit, Simulationseffizienz und Robustheit darstellt.

## 2.2. Mechanische Parameter des Stoffes

Die mechanischen Eigenschaften von Gewebematerialien begründen ihre Reaktionen auf Reize, wie Deformationen, physikalische Begrenzungen oder kräftige Muster. Eine beliebige Anzahl von Parametern kann zur Modellierung der Verhaltensweisen, die in einigen Anwendungen auftreten können, bestimmt werden. Eine Standardmenge von Parametern dagegen wird zur Darstellung der wichtigsten mechanischen Charakteristika des Gewebematerials verwendet.

### 2.2.1. Gewebestruktur

Die Gewebefasern in der Stoffoberfläche können unterschiedlich aufgebaut sein. Die Hauptstrukturen sind folgende:

- Geflochtene/gewebte Strukturen (Abb.2) sind steif, dünn und einfach herzustellen, deshalb sind sie die am meisten verwendeten Strukturen in Kleidungsstücken.
- Die gestrickte Struktur (Abb.3) ist locker und sehr elastisch, und wird für gewöhnlich als Maschenware verarbeitet.
- Die nicht-gewebte Struktur ist unstrukturiert, wie beispielsweise Papierfasern.

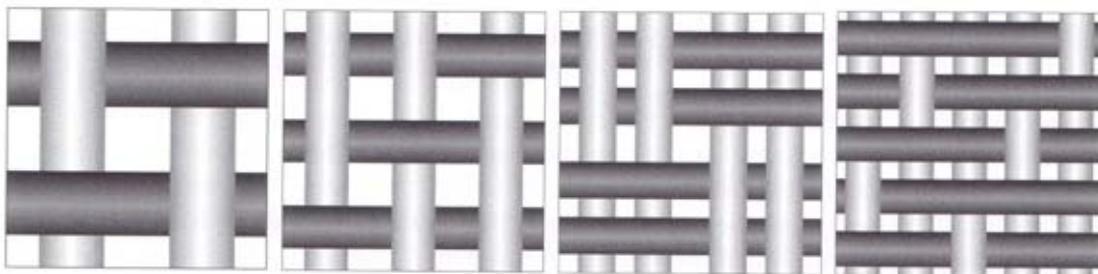


Abb.2: Gewebestrukturen

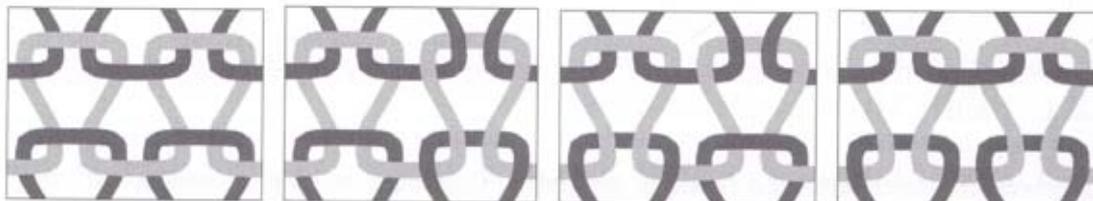


Abb.3: gestrickte Strukturen

Die Gewebestruktur beeinflusst das mechanische Verhalten der Gewebematerialien, die hauptsächlich durch folgende Eigenschaften festgelegt werden:

- Natur der Faser: Wolle, Baumwolle, Synthetik
- Fadenstruktur: Durchmesser, Innenfaser, Garnstruktur
- Fadenanordnung: gewebt oder gestrickt, teilweise Mustervariationen
- Mustereigenschaften: straff oder locker

### 2.2.2. Grundlegende mechanische Eigenschaften

Die mechanischen Eigenschaften deformierbarer Oberflächen können in vier Hauptgruppen unterteilt werden:

- Elastizität, welche die internen Kräfte charakterisiert, die aus einer gegebenen geometrischen Deformation resultieren.
- Viskosität (Zähigkeit), welche die internen Kräfte enthält, die aus einer gegebenen Deformationsgeschwindigkeit resultieren.
- Plastizität (Verformbarkeit), die beschreibt, wie sich die Eigenschaften laut dem Deformationsverdegang entwickeln.
- Strapazierfähigkeit, welche die Grenze definiert, an der die Struktur brechen würde.

#### 1. Arbeitshypothesen

Am wichtigsten sind die elastischen Eigenschaften. Die Deformationen sind oft klein und langsam genug, um die Effekte von Viskosität, Plastizität und Strapazierfähigkeit nur geringfügig erscheinen zu lassen. Die Ausrichtung des Materials hat keinerlei Einfluss auf seine mechanischen Eigenschaften (Isotropie<sup>1</sup>). Die Eigenschaften des Stoffes hängen jedoch von ihrer Ausrichtung relativ zum Gewebefaden ab.

#### 2. Elastizität

Elastische Effekte können in verschiedene Bereiche unterteilt werden. Zum einen in die metrische Elastizität, die die Deformation entlang der Oberflächenebene angibt.

---

<sup>1</sup> Isotropie = Richtungsunabhängigkeit der physikalischen und chemischen Eigenschaften von Stoffen; auch die Richtungsunabhängigkeit des physikalischen Raumes selbst

Zum anderen in Krümmungs-Elastizität, was einer Deformation orthogonal zur Oberflächenebene entspricht.

Die metrische Elastizität ist der wichtigste Aspekt der Gewebe-Elastizität und wird durch Terme der Dehnungs-Spannungs-Relation beschrieben, die folgende drei Parameter enthält:

- Young Absolutwert  $E$  (auch Elastizitätsmodul genannt), der die Reaktion des Materials entlang der Deformationsrichtung zusammenfasst.
- Poisson-Koeffizient  $\nu$ , der die Reaktion des Materials orthogonal zur Deformationsrichtung charakterisiert.
- Schermodul  $G$ , das die schiefwinkligen Reaktionen beschreibt.

Entlang der Richtung  $i$  und orthogonal zur Richtung  $j$  werden diese Relationen folgendermaßen ausgedrückt:

$$\varepsilon_{ii} = \frac{1}{E_i} \sigma_{ii} \quad \varepsilon_{jj} = \frac{\nu_{ij}}{E_i} \sigma_{ii} \quad \varepsilon_{ij} = \frac{1}{G_{ij}} \sigma_{ij} \quad (\text{F1})$$

Sie werden die Gesetze von Hook, Poisson und einfaches Schergesetz genannt.

Dabei ist  $\varepsilon$  die relative Längenänderung und  $\sigma$  die mechanische Spannung.

Stoffmaterialien sind zweidimensionale Oberflächen, für welche die

zweidimensionalen Varianten der Elastizitätsgesetze gelten. Sie sind nicht isotrop, aber die beiden orthogonalen Richtungen, die durch die Fadenläufe definiert werden, können als Hauptorientierungen für jegliche deformierbare Eigenschaft berücksichtigt werden. In dieser „orthorombischen“<sup>2</sup> Stoffoberfläche werden die beiden Richtungen „warp“ (Kettfaden) und „weft“ (Einschlagfaden) genannt und sie haben einen

besonderen Young Absolutwert,  $E_p$ ,  $\nu_p$  und Poisson-Koeffizienten,  $E_t$ ,  $\nu_t$ . Das

Elastizitätsgesetz kann wie folgt geschrieben werden:

$$\begin{bmatrix} \sigma_{pp} \\ \sigma_{tt} \\ \sigma_{pt} \end{bmatrix} = \frac{1}{1 - \nu_p \nu_t} \begin{bmatrix} E_p & \nu_t E_p & 0 \\ \nu_p E_t & E_t & 0 \\ 0 & 0 & G(1 - \nu_p \nu_t) \end{bmatrix} \begin{bmatrix} \varepsilon_{pp} \\ \varepsilon_{tt} \\ \varepsilon_{pt} \end{bmatrix} \quad (\text{F2})$$

Eine ähnliche Formel kann für die Biegungselastizität hergenommen werden, bei der allerdings der Poisson-Koeffizient für die Biegung gleich null ist. Um die Relation zwischen der Krümmungsdehnung  $\tau$  und der Spannung  $\gamma$  auszudrücken, verwendet man den Flexionsabsolutwert  $B$  und die Flexionssteifheit  $K$  und erhält dadurch die Formulierung:

<sup>2</sup> orthorombisch = ein- und ein-achsig, rombisch

$$\begin{bmatrix} \tau_{pp} \\ \tau_{tt} \\ \tau_{pt} \end{bmatrix} = \begin{bmatrix} B_p & 0 & 0 \\ 0 & B_t & 0 \\ 0 & 0 & K \end{bmatrix} \begin{bmatrix} \gamma_{pp} \\ \gamma_{tt} \\ \gamma_{pt} \end{bmatrix} \quad (\text{F3})$$

Im Zusammenhang mit linearer Elastizität werden diese Parameter meist in Studien über die Eigenschaften von Gewebe benutzt.

### **2.3. Implementierung eines mechanischen Modells**

Um Simulationen durchzuführen, benötigt man eine präzise analytische Beschreibung, die für die inneren Kräfte eines Materials Constraint-Deformations-Relationen modelliert (vgl. 2.3.1.).

Diese Relationen können dann mit den Universalgesetzen der Mechanik kombiniert werden, um Gleichheiten zu erhalten, die die Entwicklung des mechanischen Systems bestimmen (vgl. 2.3.2.).

Die Lösung dieses analytischen Systems erhält man durch Benutzung verschiedener Schemata (vgl. 2.3.3.).

#### **2.3.1. Definition der Verhaltensgesetze**

##### 1. Die mechanischen Parameter

Für die Stoffsimulation werden nur die bedeutenden Elastizitätsparameter herangezogen:

- Young-Absolutwert, der Hauptelastizitätsparameter.
- Schermodul, das die Schersteifheit ausdrückt.
- Poisson'scher Koeffizient, der für stark dehbare Stoffe bedeutend ist.
- Krümmungs-Absolutwert, der die Fähigkeit zur Faltenbildung angibt.

Die Viskositätsparameter werden genauso definiert, hängen aber von der Dehnung ab. Sie werden für die mechanische Simulation benötigt, um Stabilität zu erhalten. Nicht nur Viskosität, sondern auch Plastizität benötigt mechanische Energie. Manche stärker deformierbare Materialien kehren nicht zu ihrem anfänglichen Gleichgewichtspunkt zurück, wenn sie sehr stark verformt und dann wieder losgelassen wurden (vgl. überdehnte Feder).

## 2. Mechanisches Modellieren

Die obigen Parameter beschreiben die Kurve, die im mechanischen Simulationssystem wiederhergestellt werden muss.

Ein vereinfachtes mechanisches Modell erhält man, indem man diese Kurven auf einen einfachen mathematischen Ausdruck reduziert. Abhängig von der gewünschten Genauigkeit gibt es dafür mehrere Methoden:

- Ein lineares Modell, bei dem die Eigenschaft proportional zu der Deformation ist. Diese Modelle erfordern nur einen Koeffizienten für jede Eigenschaft und lassen sich am einfachsten anwenden. Lineare Modelle sind für gewöhnlich zutreffend, wenn die Deformationen klein bleiben.
- Ein Polynom-Modell, bei dem die Eigenschaft durch eine Funktion höherer Ordnung dargestellt und durch die entsprechenden Koeffizienten repräsentiert wird. Diese Modelle sind für Deformationen von größerer Amplitude geeignet und in Bezug auf die mathematischen Ausdrücke relativ einfach zu handhaben.
- Ein Intervall-Modell, bei dem die Kurve über erfolgreiche Intervalle angenähert wird. Diese Kurve drückt die Eigenschaft aus und wird für gewöhnlich als lineare, manchmal als Spline-Kurven höherer Ordnung gezeichnet. Diese Modelle sind geeignet für hohe nichtlineare Eigenschaften, die wichtige Unterbrechungen enthalten.
- Ein diskretes Modell für eine hohe Genauigkeit, bei dem die Kurve komplett getrennt und in einer Datentabelle gespeichert ist. Dieses ist jedoch selten von Interesse bei Stoffsimulationsmodellen.

Dabei ist die Wahl des Modells abhängig von der Genauigkeit, dem Deformationsbereich und der Art und dem Betrag der mathematischen Operation.

## 3. Umgebungsparameter

Die offensichtlichste externe Kraft, die auf den Stoff ausgeübt wird, ist die allgemeine Schwerkraft (Beschleunigung:  $9,8 \text{ m/s}^2$ ).

Aerodynamische Effekte resultieren aus der Interaktion zwischen Stoff und der umgebenden Luft. Eine ganze aerodynamische Simulation des Stoffes ist sehr schwierig zu berechnen, deswegen genügt es normalerweise, die einfachen, die

linearen, die anisotropen<sup>3</sup> und die Viskositätskräfte zu implementieren, die auf eine Stoffoberfläche ausgeübt werden.

Kleidung bewegt sich nicht unabhängig in der Luft, sondern wird getragen. Somit wird ihre Form durch den Kontakt und die Interaktion mit den darunter liegenden Oberflächen festgelegt. Diese Kräfte werden berücksichtigt, wenn es um die Kollisionsantwort zwischen simulierten Objekten geht. Kollisionskräfte schließen Reaktionskräfte mit ein, die in der Richtung orthogonal zur Kontaktfläche wirken und die gegenseitige Durchdringung der Objekte verhindern.

Es ist wesentlich, die Reaktions- und Reibungskräfte in die Kleidungssimulation einzubeziehen, falls Bewegung im Spiel ist.

### 2.3.2. Fundamentale Gesetze der Mechanik

Die Entwicklung eines mechanischen Modells, das Stoff darstellt, wird durch die fundamentalen Gesetze bestimmt, die auf das Simulationssystem angewendet werden. Diese Gesetze müssen mit den Eigenschaften, die das Verhalten des Materials charakterisieren, kombiniert werden.

#### 1. Das zweite Gesetz von Newton

$$F(t) = M \frac{d^2 P(t)}{dt^2}$$

$P$  = Position der Punktmasse abhängig von der Zeit;  
 $F$  = Summe der darauf angewendeten Kräfte (F4)

Das zweite Gesetz von Newton kann auch dazu verwendet werden, um die Vorstellung von einem Infinitesimal-Partikel zu beschreiben. Sowohl die externen, als auch die internen Kräfte, die darauf ausgeübt werden, sind das Ergebnis der Deformation des Materials.

Aus diesen Darstellungen können die Gleichungen von Lagrange abgeleitet werden.

#### 2. Erhaltungsgesetze

Abgeleitet vom Newton'schen Gesetz lassen sich Erhaltungsgesetze auf Bewegung, Drehmoment und mechanische Energie anwenden.

---

<sup>3</sup> Anisotropie = Richtungsabhängigkeit

Energieerhaltung besagt, dass sich in einem mechanischen System die innere Energie gemäß der Arbeit der externen Kräfte, die auf dieses System ausgeübt werden, entwickelt. Die Energie für ein rein mechanisches System setzt sich aus der potentiellen und der kinetischen Energie zusammen. Die potentielle ist das Ergebnis der Arbeit der inneren Erhaltungskräfte und die kinetische Energie resultiert aus der Geschwindigkeit der Materialmasse.

### 2.3.3. Definition eines Simulationsschemas

Zur Implementierung eines mechanischen Simulationssystems müssen die Verhaltensgesetze des Materials mit den mechanischen Gesetzen in einem einzigen Rahmen kombiniert werden, der auf einer angemessenen geometrischen Darstellung der zu simulierenden mechanischen Einheiten basiert.

#### 1. Numerische Simulation

Kombiniert man die Gleichungen von Materialverhalten mit mechanischen Gesetzen, so ergeben sich komplexe Systeme mathematischer Gleichungen, für gewöhnlich partielle Differentialgleichungen oder andere Arten von Differentialsystemen. Für komplexe Stoffsimulationen ist die einzig praktische Lösung die Implementierung von numerischen Methoden.

Es gibt mehrere Schemata zur Durchführung mechanischer Simulationen. Die beiden wichtigsten sind (Abb.4):

- 1) „Kontinuum-Mechanik“, die den Zustand von Materialoberflächen und Volumen untersucht, während sich die Mengen fortlaufend in Raum und Zeit ändern. Jeder physikalische Parameter des Materials wird durch einen Skalar- oder Vektorwert dargestellt, der ständig von Position und Zeit abweicht. Mechanische Gesetze können dann als eine Menge von partiellen Differentialgleichungen dargestellt werden.
- 2) „Partikel-Systeme“ unterteilen das Material selbst als eine Menge von Punktmassen („Partikel“), die mit einer Menge von „Kräften“ aufeinander wirken, welche annähernd das Verhalten des Materials modellieren.

Der Unterschied zwischen diesen beiden Schemata liegt darin, dass ein Partikel-System ein diskretes Modell ist, das auf einer ähnlich diskreten

Oberflächenrepräsentation aufgebaut ist. Die Kontinuum-Mechanik dagegen stellt ein fortlaufendes Modell dar, das dann diskretisiert wird.

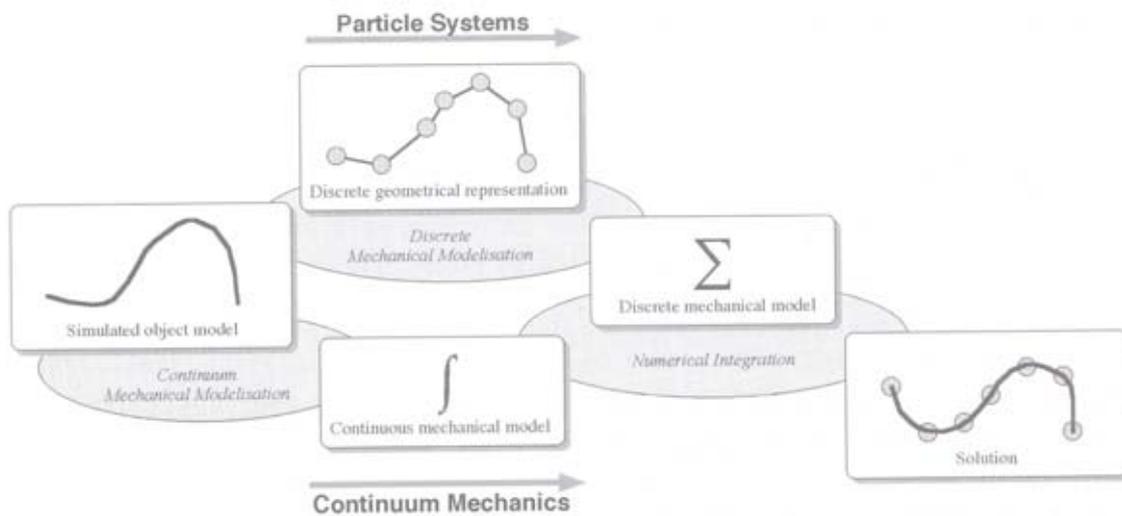


Abb.4: Schemata zur Durchführung mechanischer Simulationen

## 2. Geometrische Darstellung der Stoffoberfläche

Da es ein praktischer Weg ist, die Form und die Bewegung des Stoffes als eine kompakte und leicht zu manipulierende Menge von Daten zu beschreiben, ist die geometrische Diskretisierung notwendig. Um Stoff effizient zu simulieren, benötigt man einen gut strukturierten Rahmen zur Beschreibung seiner Geometrie.

### 2.1. Polygonale Meshes

Ein polygonales Mesh ist die einfachste Datenstruktur zur Darstellung geometrischer Oberflächen. Die Oberfläche ist unterteilt in ebene Polygone, die durch Kanten getrennt sind, die wiederum durch Scheitelpunkte verbunden sind. Je kleiner die Polygone sind, desto genauer ist das Mesh, wobei dafür aber auch mehr Daten benötigt werden, um sie darzustellen.

Es gibt mehrere Arten von polygonalen Meshes, am gebräuchlichsten sind jedoch Dreiecke oder Vierecke. Ein Mesh ist geometrisch und vollkommen regelmäßig, da seine Elemente alle die gleiche Form und Größe haben. Für unregelmäßige Meshes kann ein Grad an geometrischer Regelmäßigkeit erreicht werden, wenn man die Größe und Form zumindest grob einheitlich machen kann. Es herrscht eine Mesh-

Anisotropie vor, wenn globale, geometrische Eigenschaften abhängig von der Richtung der Mesh-Oberfläche variieren. Der wichtigste Grund für Anisotropie sind verlängerte Mesh-Elemente.

Polygonale Meshes können einfach verformt werden, indem man ihre Scheitelpunkte an die neuen Oberflächenpositionen bewegt. Eine glatte, gekrümmte Oberfläche benötigt jedoch geometrische Konsistenz und Regelmäßigkeit zwischen mehreren Scheitelpunkten.

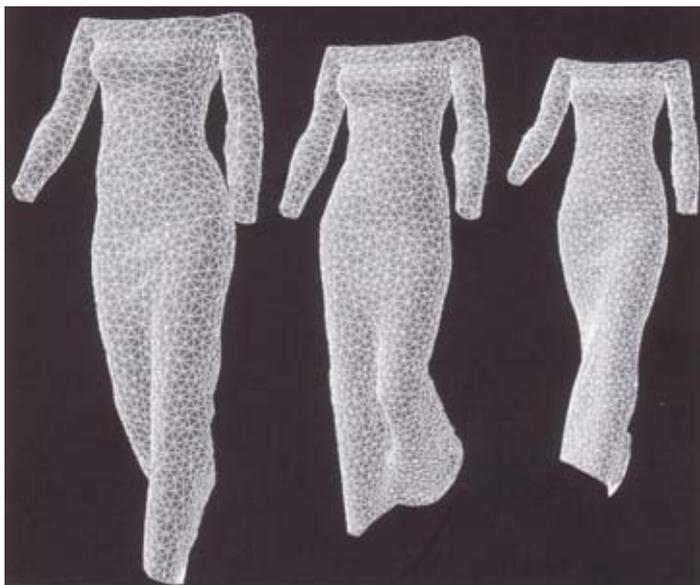


Abb.5: Ein Kleid, dargestellt als polygonales Mesh

## 2.2. Oberflächen höherer Ordnung

Oberflächen hoher Ordnung können verwendet werden, um die Menge an geometrischen Daten zu reduzieren, die zur Beschreibung glatter, gekrümmter Oberflächen notwendig sind.

Eine gekrümmte Oberfläche wird durch eine Menge von gekrümmten Patches mit einer reduzierten Anzahl von Kontrollpunkten beschrieben, welche die geometrische Information tragen. Da nur eine geringe Anzahl von Kontrollpunkten erforderlich ist, um die Form der Oberfläche zu steuern, ist es viel schwieriger, die ganze Form zu kontrollieren, da jeder Kontrollpunkt implizit eine ganze Oberflächenregion verändert.

### 2.3. Hierarchische und progressive Meshes

Zum Überwinden obiger Grenzen verwendet man hierarchische und progressive Meshes. Die beste Methode ist die Unterteilung des Meshes in Abhängigkeit von der Oberflächenkrümmung und der akzeptierbaren geometrischen Fehler der Meshdarstellung. Verschiedene „Levels of Detail“ könnten berücksichtigt werden, entweder nach oben hin, indem man größere Polygone in größere und gröbere Bereiche gruppiert, oder nach unten hin, durch Unterteilung der Meshpolygone in kleinere Bereiche.

Hierarchische Schemata können auf jede Oberflächenbeschreibung angewendet werden, entweder direkt auf polygonale Meshes oder auf Patches einer Beschreibung hoher Ordnung.

### **2.4. Mechanische Simulationssysteme**

Das Ziel einer mechanischen Simulation ist es, ein Modell zu produzieren, mit dem man die Basiseigenschaften eines Gewebes schnell, aber realistisch simulieren kann. Was immer man auch für ein mechanisches Modell auswählt, es muss eine adäquate Implementierung definiert werden – ein Algorithmus, auf dem neuesten Stand der Technik, und numerische Methoden –, um dieses Ziel zu erreichen.

#### **2.4.1. Ein gutes Simulationssystem**

Effiziente Algorithmen sind immer das Ergebnis von algorithmischen Kompromissen, welche die Qualitäten der Ergebnisse mit den Berechnungszeiten ausgleichen.

##### 1. Qualitäten

Die wichtigen Qualitäten eines guten mechanischen Simulationssystems sind:

- Funktionsumfang: Das Simulationssystem sollte das mechanische Verhalten und die zu simulierenden Eigenschaften unterstützen. Die einfachsten Modelle unterstützen lediglich einige Grundformen linearer Elastizität, während sich fortgeschrittene Modelle mit der Anisotropie und dem hohen nichtlinearen Verhalten bezogen auf Elastizität, Viskosität, Plastizität in Verlängerung oder

Krümmung befassen können. Sie enthalten fortgeschrittene aerodynamische Effekte und Möglichkeiten für verschiedene dynamische Beschränkungen.

- Genauigkeit: Das mechanische System sollte ganz genau simuliert werden. Abhängig von der Anwendung – Computergrafik oder mechanische Technik – können die Anforderungen an Genauigkeit sehr unterschiedlich sein. Das reicht von visuellem Realismus bis hin zur quantitativen Genauigkeit einiger mechanischer Zustände.
- Robustheit: Das Simulationssystem sollte so konzipiert sein, das mechanische System genau zu berechnen, auch wenn der Kontext während der Simulation variieren kann. Das Hauptproblem resultiert für gewöhnlich aus der numerischen Instabilität, die durch unzureichende Raum- und Zeitdiskretisierung in numerischen Algorithmen verursacht wird.
- Geschwindigkeit: Die Geschwindigkeit ist einer der Hauptkriterien eines Simulationssystems.

Abhängig von der Anwendung kann die Geschwindigkeitsberechnung eine Einschränkung von unterschiedlicher Wirkung werden:

- Für Offline-Berechnungssysteme, die keine Benutzerinteraktion benötigen, sind die totale Berechnungszeit und der Betrag der Zeit, die der Benutzer für das Warten auf das Ergebnis eines Simulationssystems aufbringen kann, die Bedingungen im Entwicklungsablauf.
- Für interaktive Anwendungen, in denen der Benutzer mit dem zu simulierenden Stoff interagiert, ist die Berechnungszeit zwischen 2 Frames kritischer. Die erforderliche Zeit ist begrenzt durch die vom Benutzer wahrgenommenen Effekte seiner Interaktionen.
- Für Echtzeit-Anwendungen, bei denen die Berechnungszeit strikt festgelegt ist, sollte die Geschwindigkeit der simulierten Phänomene berechnet werden.

## 2. Kompromisse und Auswahl

Es ist jederzeit möglich, ein sehr genaues und allgemeines Simulationssystem zu errichten, mit stark verfeinerter, geometrischer Darstellung, allerdings immer auf Kosten der Zeit.

## 2.1. Simulationsgenauigkeit

Es ist nicht besonders schwierig, ein genaues mechanisches Modell zu definieren, solange man die Berechnungs-Effizienz vernachlässigt. Die Schwierigkeit besteht darin, den besten Kompromiss zwischen Genauigkeit und Geschwindigkeit zu finden, so dass eine realistische Stoffanimation für die Computergrafik sehr schnell umgesetzt ist und so wenig Zeit wie möglich benötigt wird.

Ungenauigkeit bei Simulationen ist das Ergebnis verschiedener Faktoren:

- Grobheit der Oberflächen-Unterteilung: Die Topologie<sup>4</sup> und Glätte des triangulären Meshes stellt die Oberflächenform und Deformation mehr oder weniger genau dar.
- Näherungen im mechanischen Modell: Das „reale Material“ wird niemals exakt modelliert, und sein Verhalten wird durch vereinfachte mechanische Gesetze angenähert, die weder verschiedene nichtlineare Verhalten noch lokale Veränderungen in den mechanischen Strukturen entlang der Oberfläche in Betracht ziehen.
- Genauigkeit der Simulationsmethode: Der Simulationsprozess ist numerisch und iterativ, und seine Genauigkeit wird sehr stark auf seine Diskretisierungen im numerischen Modell bezogen, teilweise auf das gewählte Zeitintervall.

Der erstgenannte ist der bedeutendste Grund für die Ungenauigkeit, zum einen wegen der geometrischen Näherung und zum anderen wegen der hohen Abhängigkeit vom Verhalten des Modells.

## 2.2. Bedeutende Kompromisse

Auf der einen Seite der Kompromissmöglichkeiten steht die Geschwindigkeitsberechnung, auf der anderen Seite die Simulationsgenauigkeit.

Die Wahl des richtigen Algorithmus richtet sich nach dem Zweck der Anwendung:

- Echtzeit- und interaktive Anwendungen passen zu harten Berechnungszeit-Anforderungen und vereinfachten Modellen, für die eine schnelle Berechnung notwendig ist.

---

<sup>4</sup> Topologie = die Lehre von der Lage und Anordnung geometrischer Gebilde im Raum

- Computergestaltung und Simulationsanwendungen erfordern einige quantitative Reproduktionen von Deformationsphänomenen und somit ein passendes Modell, das genau genug ist, diese präzise zu simulieren.
- Eine Anwendung mit hohen variablen Situationen und Benutzerinteraktion erfordert ebenfalls Robustheit, um mit jeder Situation zurechtzukommen, sogar wenn sie nicht physikalisch gültig ist.

### 2.3. Berücksichtigung der Robustheit

Die wichtigste Eigenschaft eines mechanischen Simulationssystems ist die Genauigkeit: Gegeben sei ein realistisches mechanisches System und einige realistische mechanische Parameter. Das Ergebnis dieser Simulation sollte der Realität entsprechen und die erwartete Entwicklung des Modells so genau wie möglich reflektieren.

Robustheit bezieht sich vor allem auf die Fähigkeit des Modells, mit unerwarteten Situationen umzugehen, die etwas unter den Grenzen des mechanischen Modells liegen. Beispielsweise können Manipulationen durch den Benutzer auf ein geometrisches Objekt zu Deformationen führen, die dynamisch nicht korrekt sind – wie unrealistische Deformationen oder unendliche Beschleunigung. Die exakte mechanische Antwort auf diese Situationen zu berechnen, ist sinnlos. Es würden solche unrealistischen Effekte hervorgerufen werden, dass dies zu widersprüchlichen und unwiderruflichen Ergebnissen und numerischer Instabilität führen könnte.

Ein realistisches Verhalten kann also nicht erwartet werden, wenn die geometrischen Deformationen selbst nicht realistisch sind. Ein robustes System sollte in der Lage sein, extreme Situationen zu meistern, möglicherweise durch eine Änderung der Simulation weg vom originalen mechanischen Modell.

### 2.4. Überlegungen zur Verfeinerung des Meshes

Die Trennung des geometrischen Modells hängt stark zusammen mit den erwarteten Deformationen einer Oberflächenregion und teilweise mit Krümmungs-Deformationen, wie Falten und Kniffen. Eine genaue Biegung wird beispielsweise nur dann durch ein präzises mechanisches Modell genau simuliert, wenn seine geometrischen Repräsentationen mindestens ein paar dutzend Polygone über dem

bindenden Abschnitt enthalten. Alle Kniffe eines Kleidungsstücks auf diese Weise zu modellieren, würde jedoch mehrere hunderttausend Polygone für die gesamte Oberfläche erfordern, was mit derzeitigen Technologien noch nicht zu berechnen ist. Da Biege- und Kniff-Muster auf einer in einem bestimmten Zusammenhang simulierten Stoffoberfläche meist schwer vorhersehbar sind, würde ein optimiertes Mesh an diesen Zusammenhang angepasst und in der Oberflächenregion mit einer möglicherweise anisotropen Verfeinerung entlang der Krümmungsrichtung verfeinert werden, damit es kurviger ist.

#### 2.4.2. Geometrische Modelle

Geometrische Modelle stellen ein Beispiel für Zwischensimulation dar, in denen die Entwicklung eines Simulationssystems von Gesetzen und Verhalten modelliert wird, die von geometrischen Beschreibungen und ihren hergeleiteten Annäherungen abstammen. Bei geometrischen Modellen stammen die Bewegung und Deformation des Stoffes von geometrischen Krümmungen und Funktionen ab, in denen die Zeit als Parameter auftritt, so dass sie eine Anzahl von Kriterien erfüllen. Dazu gehören die Erhaltungsgesetze für Oberflächen und Moment, sowie die Kollisionseffekte mit anderen Objekten und externen Faktoren wie Schwerkraft und Wind.

Für die realistische Stoffsimulation ist hauptsächlich die Erhaltung des Oberflächenbereichs verantwortlich. Der bedeutendste visuelle Effekt ist das Verhalten der Falten. Wenn eine Oberfläche deformiert wird, sollte die metrische Länge jeder  $\text{arc}^5$ -Drehung auf der Oberfläche mehr oder weniger konstant bleiben. Die Hauptkonsequenz, die dadurch entsteht, ist das Auftreten von Falten, wenn die Oberfläche entlang einer Richtung zusammengedrückt wird.

Geometrische Modelle können diese Überlegungen enthalten, um faltige Muster auf deformierten Oberflächen zu erzeugen. Das Hauptinteresse geometrischer Simulationsmodelle ist ein Modell, das effizient zu berechnen und leicht kontrollierbar ist, und das die Simulation in verschiedene Zusammenhänge umformen kann. Der geometrisch deformierte Zustand des Stoffes kann kontrolliert werden, da er direkt aus den Simulationsparametern berechnet wird.

---

<sup>5</sup> arc = Formel-Zeichen, Bogenmaß eines Winkels; auch Bezeichnung für den (in Bogenmaß gemessenen) Polarwinkel in einem Polarkoordinatensystem, speziell für den Polarwinkel in der Gauß'schen Zahlenebene

Besonders wichtig ist der Gebrauch geometrischer Modelle in sehr speziellen Zusammenhängen, in denen Verhalten und Entwicklung bekannt sind und einfach in das Modell integriert werden können. Diese Modelle bieten hohe Kontrolle und erlauben einfaches und vorhersagbares Design von Animationssequenzen. Sie können keine hohen variablen Situationen reproduzieren. Für einfache Probleme ist dies sehr effizient. Allerdings können sehr allgemeine Anwendungen, die willkürliche Stoffformen verwenden, mechanische Zusammenhänge weniger vorhersagbarer Bahnen erfordern und nicht durch solche Modelle reproduziert werden.

### 2.4.3. Kontinuum-Mechanik-Modelle

Ein stetiges Mechanikmodell beschreibt den mechanischen Zustand eines Objekts, indem es stetige Ausdrücke verwendet, die geometrisch definiert sind. Solche Ausdrücke für deformierbare Oberflächen werden als Differential-Ausdrücke formuliert und definieren die oberflächige Deformationsenergie bezogen auf die lokale Oberflächen-Deformation – wie Dehnung, Scherung und Krümmung. Der Hauptvorteil dieser Techniken liegt darin, dass sie genaue Modelle der Materialeigenschaften vorhersehen, die direkt von mechanischen Gesetzen abstammen.

#### 1. Lagrange-Gleichungsmodelle

Die Lagrange-Gleichungsmodelle sind die Basis der stetigen Mechanik-Modelle. Sie werden zur Simulation deformierbarer Oberflächen verwendet und sind der Ausgangspunkt vieler aktueller Modelle.

Ein solches Modell berücksichtigt die Bewegungsgleichung eines infinitesimalen Oberflächenelements, die die Schwankung innere Energie – ausgelöst durch eine Partikelbewegung – ausdrückt.

Die Bewegungsgleichung wird abgeleitet von der Variationsrechnung und beschrieben in der Lagrange-Form:

$$\frac{\partial}{\partial t} \left( \mu(a) \frac{\partial r(a,t)}{\partial t} \right) + \gamma(a) \frac{\partial r(a,t)}{\partial t} + \frac{\delta \varepsilon(r)}{\delta r}(a,t) = f(r(a,t),t) \quad (\text{F5})$$

$r(a,t)$  ist die Position des Partikels  $a$ . Das Partikel des Materials hat die Koordinaten  $\alpha_1$  und  $\alpha_2$  auf der Oberfläche  $\Omega$  zur Zeit  $t$ .

$\mu(a)$  und  $\gamma(a)$  sind die Masse und Dämpfungsdichte zur Position des Partikels  $a$ .  
 $\varepsilon(r)$  ist die momentane elastische potentielle Deformationsenergie der Oberfläche.  
 Das mechanische Verhalten des Materials sollte als lokale Deformationsenergie bezogen auf die aktuelle Materialdeformation und lokal durch jeden Oberflächenpunkt ausgedrückt werden.

Zunächst sind die lokalen Deformationseigenschaften der Oberfläche zu berechnen. Die elastische Oberflächenenergie wird durch die beiden Komponenten  $G$  und  $B$  dargestellt.  $G$  wird in Ableitung von der Verlängerungs-Elastizitätsdeformation metrischer Tensor genannt.  $B$  wird aus der Krümmungs-Elastizitätsdeformation hergeleitet und als Krümmungstensor der zweiten fundamentalen Form bezeichnet.

$$G_{ij}(r(a,t)) = \frac{\partial r}{\partial \alpha_i} \cdot \frac{\partial r}{\partial \alpha_j} B_{ij}(r(a,t)) = \frac{\frac{\partial r}{\partial \alpha_1} \times \frac{\partial r}{\partial \alpha_2}}{\left| \frac{\partial r}{\partial \alpha_1} \times \frac{\partial r}{\partial \alpha_2} \right|} \cdot \frac{\partial^2 r}{\partial \alpha_i \partial \alpha_j} \quad (\text{F6})$$

Die innere Energie wird dann von diesen Ausdrücken abgeleitet. Im speziellen Fall eines linearen und isotrop elastischen Modells, in dem nur die metrische Länge und Krümmungsdeformation berücksichtigt werden, benutzt das Modell eine vereinfachte Darstellung, die die Energie folgendermaßen berechnet:

$$\varepsilon(r) = \int \sum_{\Omega, i,j=1}^2 (\eta_{ij} (G_{ij} - G_{ij}^0)^2 + \xi_{ij} (B_{ij} - B_{ij}^0)^2) d\alpha_1 d\alpha_2 \quad (\text{F7})$$

Als nächstes wird der Ausdruck

$$\begin{aligned} \frac{\partial r}{\partial \alpha_1}(m,n) &= \frac{r(m+1,n) - r(m-1,n)}{2\Delta_1} \frac{\partial^2 r}{\partial \alpha_1^2}(m,n) = \frac{r(m+1,n) + r(m-1,n) - 2r(m,n)}{\Delta_1^2} \\ \frac{\partial r}{\partial \alpha_2}(m,n) &= \frac{r(m,n+1) - r(m,n-1)}{2\Delta_2} \frac{\partial^2 r}{\partial \alpha_2^2}(m,n) = \frac{r(m,n+1) + r(m,n-1) - 2r(m,n)}{\Delta_2^2} \\ \frac{\partial^2 r}{\partial \alpha_1 \partial \alpha_2}(m,n) &= \frac{\partial^2 r}{\partial \alpha_2 \partial \alpha_1}(m,n) = \frac{r(m+1,n+1) + r(m-1,n-1) - r(m+1,n-1) - r(m-1,n+1)}{4\Delta_1 \Delta_2} \end{aligned}$$

(F8) in die Lagrange-Gleichung (F6) integriert.

Ein solches System wird nicht analytisch, sondern numerisch verarbeitet. Der günstigste Weg, das Problem in diskrete Werte umzuwandeln, ist die Benutzung eines regelmäßigen Gitters, das parallel zu den Material-Koordinaten  $\alpha_1$  und  $\alpha_2$  verläuft. Jeder Knoten des Gitters  $r(m,n)$  wird durch seine Koordinaten

$(\alpha_1, \alpha_2) = (m\Delta_1, m\Delta_2)$  bezeichnet. Die partiellen Ableitungen werden durch den Gebrauch endlicher Differenzen ausgedrückt.

Beispielabbildungen, wie Kontinuum-Mechanik-Modelle bildlich umgesetzt werden können:

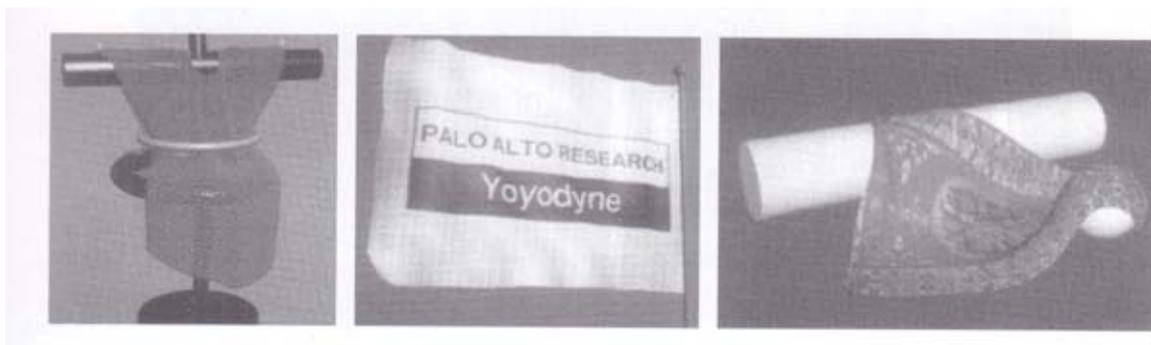


Abb.6: verschiedene Stoffsimulationen auf der Grundlage physikalischer Animation

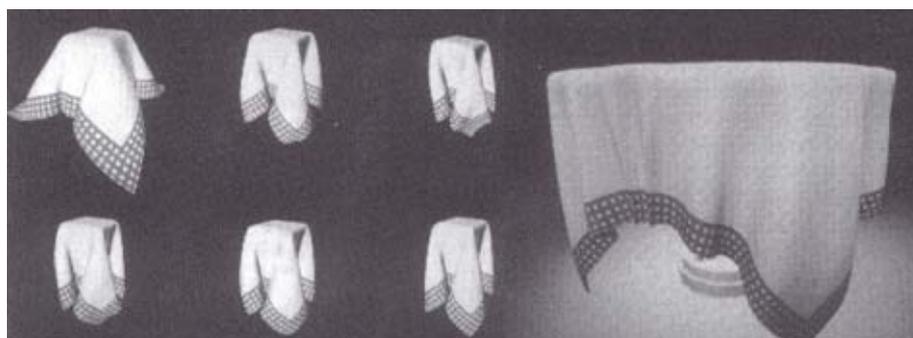


Abb.7: Drapierung von rechteckigen Stoffen (durch reguläre Meshes)

Die Abb.8 ist aus einem der ersten Filme, in der Kleidung simuliert wurde: „Flashback“. Er zeigt eine virtuelle Marilyn, bekleidet mit einem weiten Rock. Durch Wind, der aus einem Schacht unter ihr geblasen wird, ergeben sich diese Effekte. Auch die kleinen Papierfetzen am Boden werden vom Wind bewegt. Das Kleidungsstück wurde hier als eine einfache Oberfläche modelliert, dargestellt durch die Masse, die Verlängerungs- und Krümmungselastizität und die Dämpfung.

Abb.9 zeigt eine virtuelle „Modenschau“, für die die Charaktere bzw. deren Kleidungsstücke 1992 mithilfe einer Software namens MIRALab entwickelt wurden. Hier spielt nicht der Wind die entscheidende Rolle für die Bewegung der Kleidung, sondern das Laufen und die gesamte Bewegung der Charaktere.

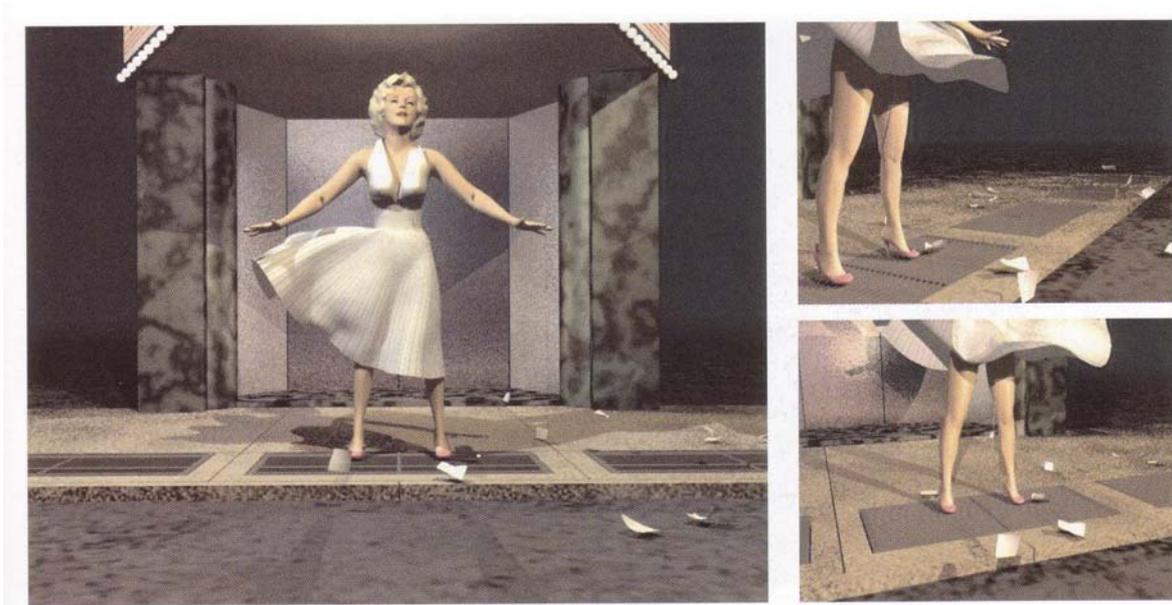


Abb.8

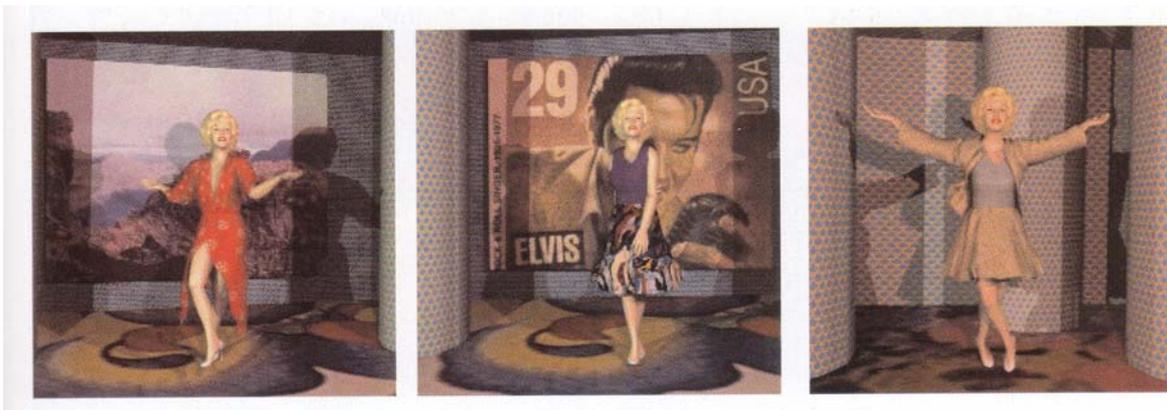


Abb.9

## 2. Finite Elemente

Finite Elemente sind ein einflussreicher Weg, stetige Mechanik-Modelle zu integrieren. Im Gegensatz zu den vorigen Modellen berechnen finite Elemente die mechanische Energie innerhalb einer vordefinierten Abtastung.

Ein diskretes Element der Oberfläche wird grundsätzlich definiert als eine interpolare Funktion über einem Patch. Diese Funktion ist von einer bestimmten Ordnung – bi-, tri- oder quadrilinear – und hat eine bestimmte Anzahl von Freiheitsgraden. Je höher die Ordnung, desto genauer passt das Element in die aktuelle Oberflächenform, umso mehr Freiheitsgrade werden aber auch benötigt.

Aus den mechanischen Eigenschaften des Materials wird die Energie berechnet, die sich auf die Deformation der Oberfläche für gegebene Werte der Interpolationsparameter bezieht.

Zur Umformung der Simulation muss ein großes lineares Gleichungssystem gelöst werden. Finite Element-Methoden sind sehr effizient für die Simulation genauer, komplexer und mechanischer Verhaltensweisen. Ihre Berechnungsmethoden jedoch sind weit von dem, was für eine schnelle Simulation in interaktiven Anwendungen an Aufwand aufgebracht werden kann, entfernt. Des Weiteren erlauben sie keine einfache und flexible Integration von Constraints, welche dynamisch entwickelt werden können.

Finite Element-Methoden waren die beste Wahl zur präzisen Simulation von elastischen Körpern in der mechanischen Technik. Die Grenzen sind jedoch offensichtlich:

Die Berechnungszeit ist extrem lang und das genaue Modellieren hoher variabler Constraints lässt sich schwer in den Formalismus finiter Elemente integrieren.

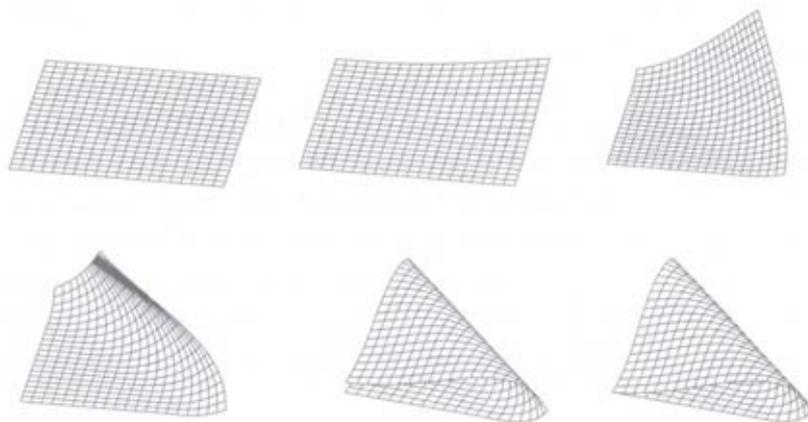


Abb.10: Simulation mit finiten Elementen

#### 2.4.4. Partikelsystem-Modelle

Anstatt die mechanischen Eigenschaften des Material-Volumens als ein Ganzes zu berücksichtigen, wäre eine andere Methode, das Material selbst als eine Menge von Punktmassen („Partikel“) zu verstehen, die mit einer Menge von „Kräften“ interagieren und so annäherungsweise das Materialverhalten modellieren.

Partikelsysteme werden verwendet, um einen großen Bereich von Phänomenen zu simulieren.

Ein günstiger Weg zum Erstellen eines Partikelsystems dieser Art ist, jeden Punkt des Meshes als ein Partikel und jede Kante oder jedes Polygon als Interaktionskräfte zwischen verschiedenen Partikeln zu berücksichtigen.

Der größte Vorteil von Partikelsystemen ist ihre Einfachheit. Es ist einfach, ein Federmassensystem zu implementieren, das effektiv die Deformation eines polygonalen Meshes, welches eine elastische Oberfläche darstellt, simuliert. Da die Positionen, Geschwindigkeiten und Kräfte jedes Partikels explizit in der Gleichung erscheinen, können nicht-lineare Effekte oder Zeitvariierende, geometrische Begrenzungen implementiert werden.

## 1. Beschreibung

Ein Partikel-System zu implementieren, ist der allgemein übliche Weg, Stoff zu animieren, der durch ein polygonales Mesh dargestellt wird. Jeder Punkt des Meshes ist ein Partikel, der eine kleine Oberflächenregion des Objekts darstellt. Ein Partikel interagiert mit Nachbarpartikeln mechanisch auf verschiedene Weisen.

### 2.1. Prinzipien

Der gewöhnliche Weg, ein mechanisch basiertes Partikelsystem numerisch zu simulieren, ist, das 2. Gesetz von Newton direkt für ein Masse-Partikel über alle Partikel zu integrieren:

$$F(t) = M \frac{d^2 P}{dt^2} \quad (\text{F9})$$

Dabei ist  $P$  die Partikelposition,

$F$  die Summe der Kräfte, angewendet auf das Partikel und  $M$  seine Masse.

Die Kräfte, die auf jedes Partikel ausgeübt werden, hängen vom aktuellen Zustand des Systems ab, der durch die Position und Geschwindigkeit aller Partikel charakterisiert wird. Diese Kräfte stellen für gewöhnlich alle mechanischen Effekte auf das System dar: innere Elastizität und Viskositätskräfte, Schwerkraft und aerodynamische Effekte und verschiedene Arten anderer externer Beschränkungen.

Einige Arten geometrischer Begrenzungen können jedoch auch geometrisch integriert werden, indem man direkt die Position oder die Geschwindigkeit des Partikels ändert.

## 2.2. Partikelsystem-Darstellungen

Das mechanische Verhalten des Materials wird als Interaktion zwischen Partikeln implementiert. Diese Interaktion ist begrenzt auf Nachbarpartikel zur Simulation von grundlegenden Elastizitätsverhalten.

Feder-Massen-Systeme sind der einfachste Weg, ein Volumenmodell zu erstellen, das ein Partikelsystem benutzt. Bei dieser Methode stellt jedes Partikel eine Punkt-Masse dar, die mit ihren Nachbarn durch eine „Feder“ verbunden ist. Die Federn stellen das elastische Verhalten des Materials dar und tendieren dazu, die Partikel an ihren anfänglichen Ruhepositionen zu lassen.

Für die Darstellung der gewöhnlichen mechanischen Parameter gibt es verschiedene Arten von Federn (Abb.11).

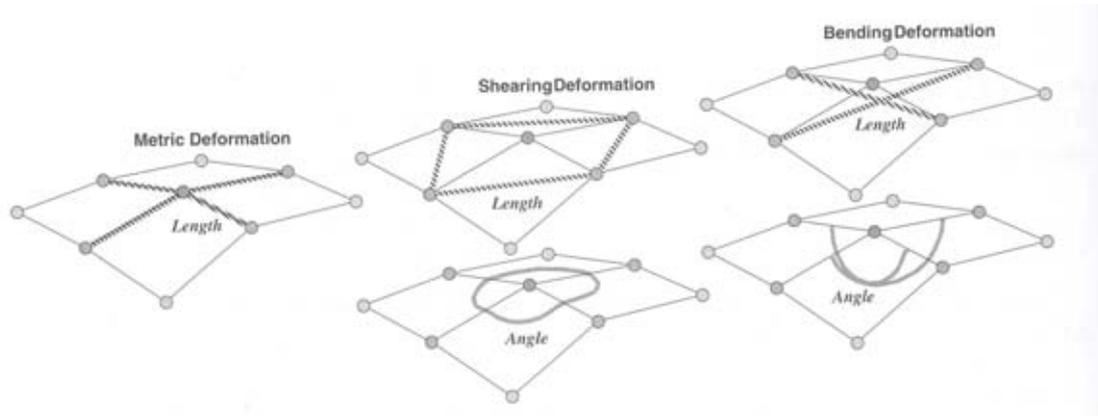


Abb.11: Verwendung der Längen oder Winkel beim Messen von Deformationen

Bei der Verwendung regulärer Gitter können metrische Elastizitäten, Scher- und Biegeelastizitäten sowohl mit Verlängerungsfedern als auch mit Beugungsfedern modelliert werden. Die metrische Elastizität wird für gewöhnlich durch Verlängerungsfedern entlang der Gitterkanten definiert. Die Scherelastizität dagegen wird entweder durch die Gitterwinkelfedern oder durch die diagonalen Verlängerungsfedern modelliert.

Auf die gleiche Weise kann die Krümmungselastizität entweder durch Beugungsfedern zwischen entgegengesetzten Kanten oder durch Verlängerungsfedern zwischen entgegengesetzten Punkten definiert werden. Die Verwendung von Verlängerungsfedern vereinfacht das Modell, da diese einfache Art von Federn leicht modelliert werden kann. Allerdings ist dies für starke Deformationen zu ungenau.

Beugungsfedern können die Scher- und insbesondere die Krümmungselastizität genauer modellieren. Das erfordert jedoch intensivere Berechnungen und benötigt Informationen über die Oberflächenorientierung, die nicht immer präzise berechnet werden kann.

Beispiele von Drapierungen, für die Partikelsysteme verwendet wurden:

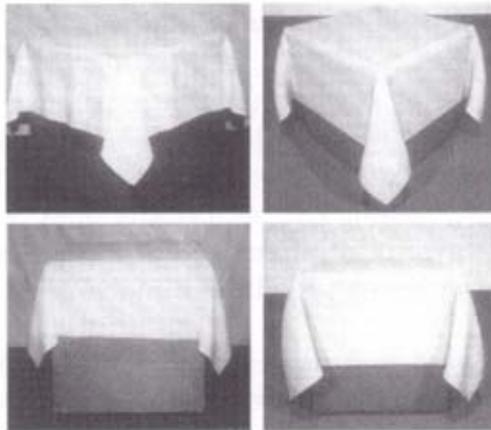


Abb.12: Die linken beiden Bilder sind eine reale Darstellung einer Tischdecke. Die rechten beiden Bilder sind eine virtuelle Darstellung mithilfe einer Partikelsimulation.

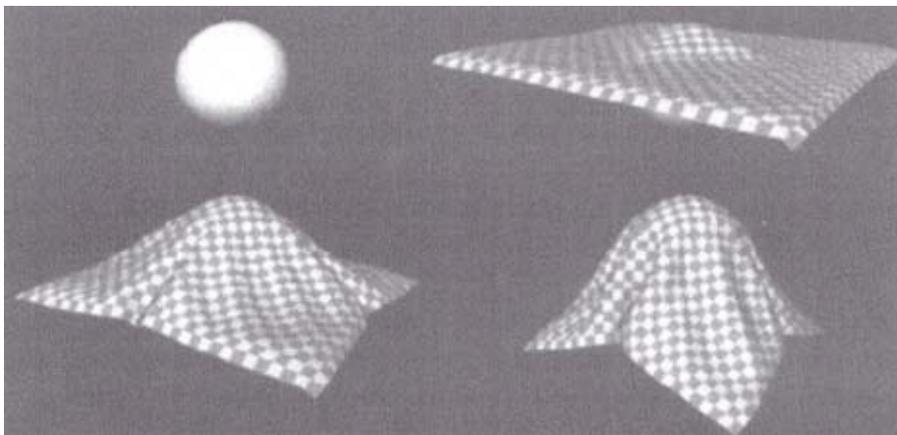


Abb.13: Stoffdrapierung mithilfe eines Partikelsystems

### 2.4.5. Ein schnelles Partikel-System für irreguläre Meshes

Partikelsysteme sind Hauptkandidaten für Stoffsimulationssysteme, da sie schnell und vielseitig sind, Implementierungen von verschiedenen Verhaltensgleichungen erleichtern und direkten Zugriff auf den Systemzustand erlauben, um Constraints und Manipulationen umzusetzen.

Es gibt viele Partikelsysteme, die in der Literatur beschrieben werden. Diese implementieren genaue, elastische, lineare Modelle, deren Merkmale präzise Elastizitäts-, Viskositäts- und Plastizitätsparameter sind. Es stellt sich hier die Frage, wie genau man wirklich sein muss.

Das einfache Federmassenmodell ist sicherlich das einfachste und schnellste mechanische Modell gestützt durch ein Partikelsystem. Die Oberfläche ist in einfache Massen unterteilt, die an ihre jeweiligen Nachbarn durch Federn verbunden sind. Die Ruhelänge der Federn befindet sich an der Originalposition der Oberfläche.

In diesem Kapitel ist ein angepasstes Partikelsystemmodell ausführlich beschrieben, das ganz genau die wichtigsten elastischen Parameter – wie den Young-Absolutwert oder den Poisson-Koeffizienten – der elastischen Oberfläche darstellen kann.

#### 1. Das einfache Federmassen-System: Effizienz und Schwäche

Das einfache Federmassen-System ist das einfachste und schnellste Partikelsystem, das für mechanische Simulation elastischer Oberflächen implementiert werden kann. Dieses Modell reicht jedoch offensichtlich nicht für die Stoffanimation, da es unrealistische Deformationen hervorruft.

Die sichtbarsten Ergebnisse dieser Artefakte sind die teilweisen oder kompletten Verluste des transversalen Oberflächenfaltens während der hohen Deformation. Ein Federmassenmodell ist jedoch numerisch stabil: es hat keine Konfigurationen, in denen einige Federkräfte aus einer Nulldivision resultieren. Ein solches Modell hat die Fähigkeit, numerische Genauigkeit wiederherzustellen.

#### 2. Verbesserung des Federmassenmodells

Das Modell vermeidet die Nachteile des einfachen Federmassenmodells, aber bleibt einfach genug, um in ein schnelles Simulationssystem implementiert zu werden.

## 2.1. Elastizität in der Ebene

Gegeben sei ein Dreieck  $(P_a, P_b, P_c)$ , in welchem durch die Deformationen die Kanten von der Ruhelänge  $(L_a, L_b, L_c)$  auf die aktuelle Länge  $(l_a, l_b, l_c)$  verlängert wurden:

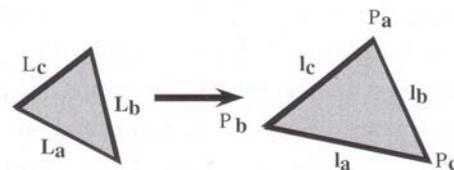


Abb.14: Deformierung eines Dreiecks: links das undeformierte, rechts das aktuelle

Die aktuellen Kantenlängen sind folgendermaßen definiert:

$$l_i = \|P_j - P_k\| \quad \begin{array}{l} i = (a, b, c) \\ j = (b, c, a) \\ k = (c, a, b) \end{array} \quad (\text{F10})$$

Des Weiteren wird der Cosinus des aktuellen Winkels berechnet  $(c_a, c_b, c_c)$  und ein weiterer konstanter Parameter  $\alpha = 1$  eingeführt.

$$\cos c_i = \alpha \frac{l_j^2 + l_k^2 - l_i^2}{2l_j l_k} \quad (\text{Cosinus-Satz}) \quad (\text{F11})$$

Im einfachen Federmassenmodell produzieren die drei Kanten eine Kraft proportional zu ihrer Verlängerung der Ruhelänge. Das System ist im Gleichgewicht, wenn die drei Kanten ihre Ruhelänge erreichen.

Die erwartete Verschiebung  $d_i$  in jede Kantenrichtung wird aus der Verlängerung der Kante in ihre entsprechende Richtung berechnet:

$$d_i = l_i - L_i \quad i = (a, b, c) \quad (\text{F12})$$

Die Federkräfte werden dann direkt aus der Verschiebung berechnet:

$$F_i = \frac{R_i}{L_i} d_i = \frac{R_i}{L_i} (l_i - L_i) \quad (\text{F13})$$

Dieses Modell lässt sich ganz leicht berechnen, reflektiert aber nicht die aktuellen Kräfte, wenn ein volles Material-Dreieck deformiert wird, da es nur auf den Dreieckskanten gelöst wurde.

Jede deformierte Kante wird eine Kraftkomponente entlang ihrer Richtung produzieren. Der daraus resultierende Effekt ist eine zusätzliche orthogonale

Deformation zu der, die vom Poisson'schen Koeffizienten hervorgerufen wird. Dadurch entstehen unrealistische Effekte, vor allem wenn die Dreiecke nicht gleichseitig sind. Eine hohe Komprimierung führt nicht zu hohem Deformationswiderstand in der Komprimierungsrichtung, sondern eher entlang der orthogonalen Richtung, an der die Kanten parallel werden.

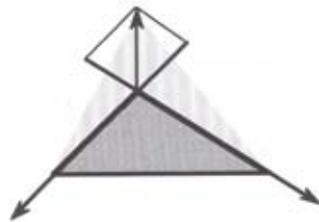


Abb.15: Eine vertikale Kompression streckt das Dreieck horizontal

Die Hauptidee des verbesserten Modells ist es, den individuellen Verlängerungsbeitrag von jeder Kante des Dreiecks zu berechnen. Dabei muss man die gegenseitige Abhängigkeit der Verschiebungen berücksichtigen, die von jeder Kante in ihre jeweiligen Richtungen generiert werden. Dadurch produzieren die kombinierten Effekte der Kantenkräfte eine genauere Grenzsituation.

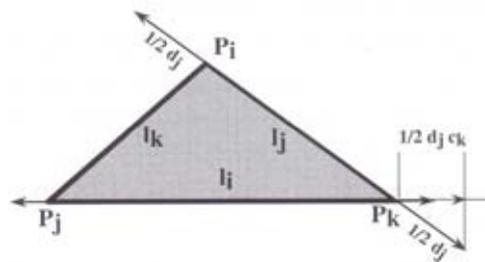


Abb.16: Berechnung des Verlängerungseffekts der Kante  $J$  zur Kante  $I$

$$d_i + \frac{1}{2}c_k d_j + \frac{1}{2}c_j d_k = l_i - L_i \quad \begin{array}{l} i = (a, b, c) \\ j = (b, c, a) \\ k = (c, a, b) \end{array} \quad (\text{F14})$$

Die Lösung des drei Gleichungs-Systems mit drei Unbekannten sieht dann folgendermaßen aus:

$$F_i = \frac{R_i}{L_i} d_i = \frac{R_i}{L_i} \frac{(c_i^2 - 4)(l_i - L_i) - (c_i c_j - 2c_k)(l_j - L_j) - (c_i c_k - 2c_j)(l_k - L_k)}{c_i^2 + c_j^2 + c_k^2 - c_i c_j c_k - 4} \quad (\text{F15})$$

Diese Voraussetzung wurde experimentell geprüft, um nicht zu viele Effekte auf das Verhalten des simulierten elastischen Materials zu haben, wie es in den meisten gewöhnlichen Situationen der Stoffsimulation der Fall ist.

Zum Schluss wird der Beschleunigungsbeitrag aus dem Dreieck, angewendet auf seine Punkte, aus den Gleichungen (F11) und (F15) oder durch folgende Gleichung berechnet:

$$P_i'' = M_i^{-1} \left( \frac{F_j}{l_j} (P_k - P_i) + \frac{F_k}{l_k} (P_j - P_i) \right) \quad \begin{array}{l} i = (a, b, c) \\ j = (b, c, a) \\ k = (c, a, b) \end{array} \quad (\text{F16})$$

## 2.2. Stabilitätskontrolle und Poisson'scher Koeffizient

Der  $\alpha$  - Koeffizient, der einen linearen Interpolationsfaktor zwischen einem einfachen Federmassensystem ( $\alpha = 0$ ) und dem oben beschriebenen Modell ( $\alpha = 1$ ) darstellt, wurde eingeführt, um jede unendliche Kraftkonfiguration auszuschließen.

Um zu verhindern, dass jede Positionskonfiguration zu unendlichen Kräften führt, setzt man  $|\alpha| < 1$ .

Wenn  $\alpha = 1$  ist, simuliert das Modell ein Gewebematerial mit einem Poisson'schen Koeffizienten gleich 0. Der Poisson'sche Koeffizient ist abhängig von der Form der Dreiecke. Zusätzlich zur Stabilität erlaubt  $\alpha$ , den Poisson'schen Koeffizienten des simulierten Gewebes zu kontrollieren.

## 2.3. Implementierung des verbesserten Modells

Der Vorteil des Modells ist nicht nur die Einfachheit, sondern auch die Fähigkeit, realistisch elastische Kräfte sogar in unregelmäßigen Dreiecksmeshes zu berechnen. Das erfordert sehr wenige Vektorberechnungen; es werden direkt Kraftbeiträge entlang der Kantenrichtung berechnet, ohne dass man ein lokales Koordinatensystem benötigt.

Das Modell kann einfach implementiert werden, es hat den gleichen Rahmen wie das einfache Partikelsystemmodell. Eine Extraberechnung muss lediglich für die Gleichungen (F11) und (F15) durchgeführt werden.

Diese Art von Modell ist ein guter Ersatz für einfache Feder-Massen-Modelle in Simulationssystemen.

## 2.4. Arbeiten mit inversen Massen

Dynamik erfordert für gewöhnlich die Masse des Partikels, der berücksichtigt werden soll. Dazu benötigt man das Gesetz von Newton, das die Beschleunigung eines Partikels anhand der Division der ganzen angewendeten Kraft auf diesen Partikel durch seine Masse berechnet.

Ein Partikel mit einer Nullmasse hätte eine unendliche Beschleunigung. Ein Partikel mit unbegrenzter Masse ist wichtig für das Modellieren von unveränderlichen Partikeln ohne Beschleunigung.

Wenn man mit der inversen Masse in der Mechanik rechnet, können benötigte Partikel einfach dargestellt werden als wären die inversen Massen gleich null. In der Beschleunigungsberechnung verwandelt sich die Massendivision in die inverse Massenmultiplikation, da diese effizienter berechnet werden kann und nie Singularwerte in die Berechnung einführt. In allen Formeln dieses Kapitels können benötigte Partikel einfach gehandhabt werden, indem man  $M_i^{-1} = 0$  setzt.

Inverse Massen werden für implizite Simulationssysteme benötigt.

## 3. Krümmungselastizität

Das bisher beschriebene Modell handelt nur von der Elastizität in der Ebene. Es muss aber noch eine andere Art von Elastizität berücksichtigt werden, die Oberflächen-Krümmung und –Beugung. Dadurch werden zwar nur schwache Effekte erzielt, lokale Deformationen jedoch schon.

Für ein perfektes anisotropes Material sollte die Oberfläche eine gewisse Krümmungselastizität für die Oberflächendicke zeigen: lokale Verdichtung und Verlängerungsdehnung treten an den inneren und äußeren Seiten der gekrümmten Oberfläche auf und verursachen somit Krümmungsspannung.

### 3.1. Krümmungsantwort

Um die krümmungselastische Antwort zu berechnen, müssen Kräfte produziert werden, die sich der Oberflächenkrümmung widersetzen. Ein Krümmungskraftmoment, das auf eine Oberfläche wirkt, muss in den gekrümmten Gebieten der Oberfläche entstehen. Die Formulierung eines solchen Effekts mag in einem stetig-mechanischen Modell einfach sein, in denen die Ableitungen zweiter Ordnung effizient Krümmungen messen, aber seine Implementierung in diskreten Meshes ist problematischer. Ein grobes Mesh kann eine höchstgekrümmte Oberfläche nie genau beschreiben. Die enthaltenen Oberflächendehformationen, die in den gekrümmten Regionen auftreten, machen jeden Versuch, die Oberflächenkrümmung zu messen, ungenauer.

Wenn Kantenwinkel zwischen Mesh-Polygonen bedeutend werden, sollte jede Krümmungsmessung nur als Annäherung an die wirkliche Durchschnittskrümmung in diesem Bereich berücksichtigt werden. Um die Krümmung in einem unregelmäßigen Dreiecksmesh zu messen, muss der Winkel  $\Theta$  zwischen zwei Dreiecken ( $ACB$ ) und ( $BDA$ ) über der Kante ( $AB$ ) berücksichtigt werden (Abb.17).

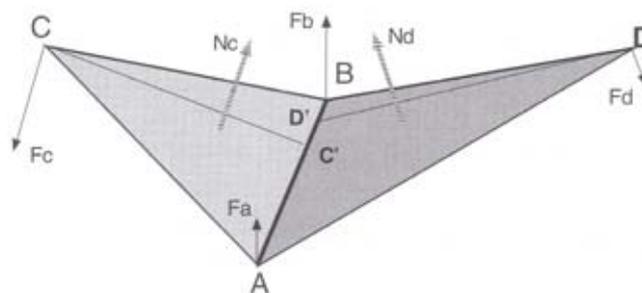


Abb.17: Berechnung der Krümmungskräfte der Punkte, die eine Kante in einem polygonalen Mesh umgeben

Das resultierende Krümmungskraftmoment ist dann gegeben durch die Kräfte  $F_a, F_b, F_c, F_d$ , jeweils angewendet auf die Punkte  $A, B, C, D$ . Der Winkel  $\Theta$  lässt sich durch seinen Sinus und Cosinus unter Berücksichtigung der Dreiecksnormalenvektoren  $\vec{N}_c$  und  $\vec{N}_d$  gemäß folgender Berechnung ermitteln:

$$\begin{aligned} \cos(\Theta) &= \vec{N}_c \circ \vec{N}_d \\ \sin(\Theta) &= \frac{(\vec{N}_c \times \vec{N}_d) \circ (\vec{B} - \vec{A})}{|\vec{B} - \vec{A}|} \end{aligned} \quad (\text{F17})$$

Um die Berechnung zu erleichtern, sollten diese trigonometrischen Funktionen nicht erweitert werden. Aus diesen Werten berechnet man das benötigte Kraftmoment  $M(\Theta)$ .  $F_c$  und  $F_d$  generieren gegenüberliegende Kraftmomente um die Kanten herum, deren globaler Effekt es ist, zwei angrenzende Dreiecke „auseinanderzufalten“. Um ihren Moment um die Achsen  $(AB)$  auszudrücken, müssen sie die folgenden Ausdrücke erfüllen:

$$\frac{\left(\vec{C} - \vec{C}'\right) \times \vec{F}_c \circ \left(\vec{B} - \vec{A}\right)}{\left|\vec{B} - \vec{A}\right|} = M(\Theta)$$

$$\frac{\left(\vec{D} - \vec{D}'\right) \times \vec{F}_d \circ \left(\vec{A} - \vec{B}\right)}{\left|\vec{A} - \vec{B}\right|} = M(\Theta)$$
(F18)

$F_c$  und  $F_d$  sollten in eine Linie mit jeweiligen Dreiecksnormalen  $\vec{N}_c$  und  $\vec{N}_d$  gebracht werden:

$$F_c = \frac{M(\Theta)}{\left|\vec{C} - \vec{C}'\right|} \vec{N}_c$$

$$F_d = \frac{M(\Theta)}{\left|\vec{D} - \vec{D}'\right|} \vec{N}_d$$
(F19)

Die zwei Punktkräfte  $F_c$  und  $F_d$  benennen das benötigte Kraftmoment. Es stellt die Krümmungselastizität dar, die sich der Beugung der Polygone um eine Kante herum widersetzt.

$F_a$  und  $F_b$  angewandt auf  $A$  und  $B$  müssen um die berücksichtigten Kanten herum mit den Erhaltungsgesetzen übereinstimmen. Der wichtigste Faktor ist das Gleichgewicht aller Kräfte, für das gilt:  $F_a + F_b + F_c + F_d = 0$ . Zusätzliche

Einschränkungen, für den totalen Nullkraftmoment, leiten zu folgenden Formeln:

$$F_a = \frac{\left(\vec{C}' - \vec{B}\right) \circ \left(\vec{B} - \vec{A}\right)}{\left|\vec{B} - \vec{A}\right|^2} F_c + \frac{\left(\vec{D}' - \vec{B}\right) \circ \left(\vec{B} - \vec{A}\right)}{\left|\vec{B} - \vec{A}\right|^2} F_d$$

$$F_b = \frac{\left(\vec{C}' - \vec{A}\right) \circ \left(\vec{A} - \vec{B}\right)}{\left|\vec{A} - \vec{B}\right|^2} F_c + \frac{\left(\vec{D}' - \vec{A}\right) \circ \left(\vec{A} - \vec{B}\right)}{\left|\vec{A} - \vec{B}\right|^2} F_d$$
(F20)

Die Berechnung dieses Ausdrucks kann mithilfe von Zwischenwerten, die bereits während der Festsetzung von  $\vec{C}'$  und  $\vec{D}'$  kalkuliert wurden, vereinfacht werden.

### 3.2. Nichtlinearität und Winkelumkehrungen

Die Berechnung des Kraftmoments  $M(\Theta)$  aus dem Kantenwinkel  $\Theta$  sollte sowohl auf der Materialkrümmungselastizität  $\Gamma$  beruhen, als auch auf dem „Formfaktor“  $\Psi$ . Letzterer berücksichtigt die Konfiguration und die Masse der Punkte, die das Dreieck umgeben.

Lineare Krümmungselastizität würde bedeuten:

$$M(\Theta) = \Gamma \Psi \Theta \quad (\text{F21})$$

(Krümmungskraftmoment proportional zum Kantenwinkel)

Dies ist jedoch eine unpraktische Formel: erstens benötigt sie die genaue Berechnung des Winkels  $\Theta$  und zweitens rendert sie keine angemessene Antwort für große Krümmungen, was der Fall ist, wenn  $\Theta$  nahe  $\pm\pi$  geht.

Das Ziel sollte eine robuste Krümmungsantwort sein, die eine Oberflächenkreuzung verhindert. Eine genaue Gewebekrümmungsantwort für große Winkel zu modellieren, ist nicht sehr sinnvoll, da diese Winkel nur eine grobe Annäherung an die tatsächliche Oberflächenkrümmung sind.

Das Grenzverhalten von  $\Theta$  nahe  $\pm\pi$  verdient besondere Betrachtung: Es reflektiert die Situation, an der die Oberfläche an eine U-ähnliche Form gebunden ist.

Normalerweise würde eine Selbstkollision weitere Krümmung an Stellen verhindern, wo die Oberfläche möglicherweise entlang der betroffenen Kanten „kreuzt“ (Abb.18).

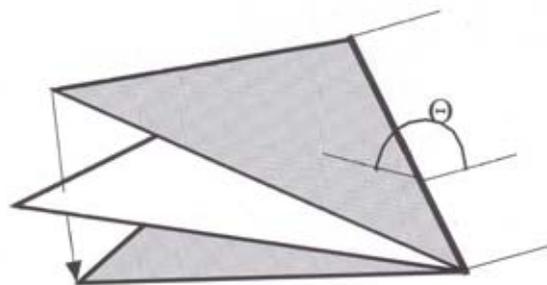


Abb.18: Krümmungskreuzen, das bei einer Kante auftreten kann

Die Situation kann jedoch nicht in ein gewöhnliches Selbstkollisionsantwortschema integriert werden, da die Flächen nicht geometrisch kollidieren und die Geometrie um die „gekreuzten“ Kanten herum konsistent bleibt.

Zwei verschiedene Lösungen kommen in Betracht:

- 1) Wenn eine Krümmungskreuzung auftritt, wird der Winkel  $\Theta$  kleiner  $\pm\pi$  und  $|\Theta| > \pi$ . Da der Winkel  $\Theta$  nur geometrisch im Intervall  $[-\pi, \pi]$  gemessen wird, muss ein Trackingsystem zur Addition oder Subtraktion von  $2\pi$  Termen implementiert werden, damit der aktuelle Winkel erreicht werden kann.
- 2) Der einzige Winkel, der direkt im Intervall  $[-\pi, \pi]$  gemessen wird, ist der Winkel  $\Theta$ , somit ist jedes Kreuzen vergessen. Um das Vorkommen von Kreuzungen zu begrenzen, muss die Antwort große Werte annehmen, wenn der Winkel  $\Theta \pm\pi$  erreicht.

Theoretisch ist erstere die beste Lösung, da sie sicherstellt, dass die globale geometrische Meshkonfiguration an ihren Anfangszustand zurückkehrt. Praktisch ist das nicht wahr, da Selbstkollision tatsächlich erst in den Nachbarschafts-Elementen auftritt. Entweder verhindern diese Selbstkollisionen das Kreuzen oder, wenn ein Kreuzen bereits auftrat, das Rückkreuzen. Die zweite Lösung wird also bevorzugt, da nur eine passende Antwort berechnet werden muss.

Um ein ähnliches lineares Verhalten wie in der Gleichung (F21) für kleine Winkel zu erhalten, führt man den Parameter  $g$  zum Einstellen der Höhe der Spannungswand ein und konstruiert die Antwort folgendermaßen:

$$M(\Theta) = \Gamma\Psi \frac{(1+2g)\sin(\Theta)}{1+g(1+\cos(\Theta))} \quad (\text{F22})$$

Abb.19 zeigt, wie diese Funktion verläuft. Dabei steht jede Kurve für einen anderen Wert des Parameters  $g$ .

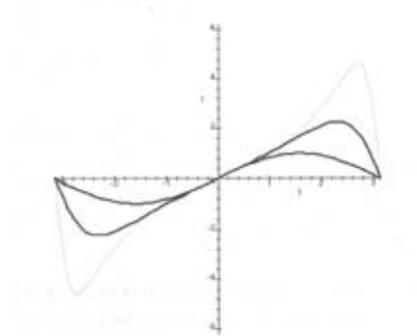


Abb.19: Krümmungsantwort des Kraftmoments

Da der Winkel wächst, wenn  $\pm\pi$  erlangt wird, erreicht die Antwort ein Maximum. Dessen Wert wird anhand der Durchschnittswerte des Parameters  $g$  angepasst und

funktioniert wie eine Kraft, die von einer Spannungswand abgeleitet wurde. Für die numerische Stabilität muss jedoch etwas an Antwort gewährleistet werden, sogar wenn eine Kreuzung durch den  $\pm \pi$ -Grenzwert auftritt, an dem die Antwort gegen null geht. Für praktische Implementierungen werden nur die sin- und cos-Winkel-Werte aus der Formel (F17) berechnet.

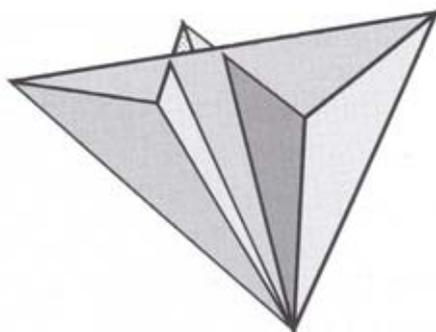


Abb.20: lokales Krümmungskreuzen, das an einem Punkt auftreten kann

### 3.3. Krümmung und Elastizität im polygonalen Mesh

Polygonale Meshes zur Beschreibung höchstdeformierter Oberflächen können Artefakte einführen, die die Renderingqualität und die mechanische Simulation, die auf dem Mesh basiert, stören.

In größerem Umfang als die Form-Deformation, beeinflussen diese Artefakte die genaue Modellierung der Krümmungselastizität für große Deformationen. Ein Problem tritt meist bei unregelmäßigen Dreiecksmeshes auf, welche mehr oder weniger deformierbar sind.

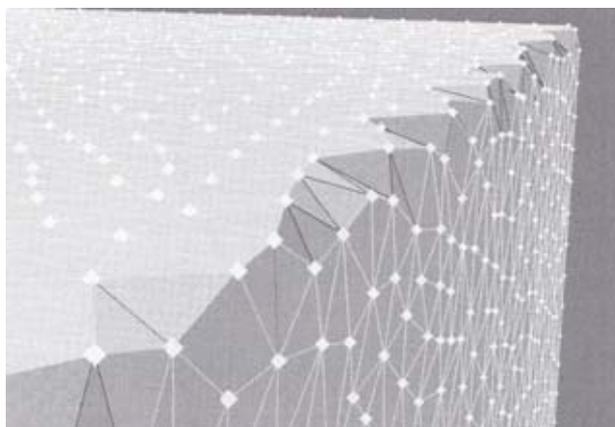


Abb.21: Eine starke Meshkrümmung generiert eine Verdichtung (siehe dunkle Kanten)

Dieser Wechsel in der Kantenlänge mag geringfügig sein, wenn der Krümmungsradius, verglichen mit der Größe der Mesh-Elemente, groß ist. Allerdings ist eine hohe Grad-Krümmung wahrscheinlich, um die Kanten, welche die gekrümmte Region orthogonal kreuzen, zusammenzudrücken, da die parallele Kantenlänge in etwa unverändert bleibt. Als Konsequenz wird die Kantenelastizität, die zum Modellieren der Elastizität in der Ebene benötigt wird, die Krümmungselastizität negativ beeinflussen.

Bei einem Modell, das die gleiche Skalierung der Oberflächenkrümmung und der Elemente hat, kann eine polygonale Winkelkontrolle nicht einfach zur Krümmungselastizität führen. Dieses Problem ist wichtig für Modelle mit hoher Steifheit in der Ebene, die die Krümmungssteifheit stört. Eine Änderung der Kantenruhelänge gemäß den geschätzten Krümmungen, gemessen an ihren Endpunkten, würde dies nahezu ausgleichen. Diese Methode erfordert jedoch für eine teilweise Genauigkeit nicht-triviale geometrische Berechnungen.

## **2.5. Numerische Integration**

Die mechanischen Modellgleichungen können nicht analytisch gelöst werden, sondern die Simulation muss zur Nutzung eines numerischen Prozesses berechnet werden. Die Implementierung effizienter numerischer Techniken ist eine Schlüsselfrage für ein effizientes Simulationssystem. In einigen Modellen, die von der stetigen Mechanik abgeleitet werden, ist die numerische Lösung in das Modell eingebettet (z.B. finite Elementmethoden). Andere Methoden wenden ein Diskretisierungsschema an, um das analytische System in eine Menge von Gleichungen umzuwandeln, die numerisch gelöst werden müssen.

Bei Partikelsystemmodellen werden der Systemzustand, die Entwicklung durch die Position und die Entwicklung der Partikel beschrieben. Die Diskretisierung der mechanischen Gesetze, die auf die Partikel wirken, produzieren normalerweise ein System von gewöhnlichen Differentialgleichungen, die numerisch entlang einer Zeitentwicklung gelöst werden müssen.

Außer bei mechanischen Systemen mit ein oder zwei Freiheitsgraden und linearen mechanischen Modellen ist es unmöglich, die differentiellen Gleichungssysteme, die die Entwicklung der mechanischen Systeme beschreiben, analytisch zu lösen. Das

numerische Ergebnis nähert sich der Lösung mithilfe einer Extrapolation von Zeitpunkt zu Zeitpunkt, indem es die Ableitungen als Entwicklungsinformation nutzt. Nachteilig bei dieser numerischen Simulation ist der „Simulationsfehler“, der bei jedem Zeitschritt größer wird. Optimierte numerische Methoden können die numerische Berechnung durchführen, indem sie effizient den Fehler minimieren. Das Hauptaugenmerk dieses Kapitels liegt auf der Teiltechnik, die direkt mit dem Partikelsystem genutzt werden kann, das im vorigen Kapitel beschrieben wurde.

### 2.5.1. Integrationstechniken

Differentielle Gleichungssysteme zu lösen, ist ein weitreichendes Thema in der numerischen Analyse. Für die mechanische Simulation, die Partikelsysteme verwendet, kann das Problem für gewöhnlich auf die Lösung des normalen Differentialgleichungssystems zweiter Ordnung reduziert werden. Dabei sind die Variablen gleich den Partikelpositionen entlang der Entwicklungszeit.

#### 1. Modellierung wie ein System erster Ordnung

$P(t)$  sei der Positionsvektor des Gesamtsystems zur Zeit  $t$ , dessen Größe der dreifachen Anzahl von Punkten entspricht.  $P'(t)$  und  $P''(t)$  sind – bezogen auf  $t$ , die erste und zweite Ableitung – der Geschwindigkeits- und Beschleunigungsvektor. Die Rolle der numerischen Integration ist es, den Zustand des Systems nach einem gegebenen Zeitpunkt  $dt$  zu berechnen. Die intuitive Methode ist, zwei erfolgreiche Integrationen durchzuführen:

$P(t + dt)$  aus  $P(t)$  und  $P'(t)$  berechnen und

$P'(t + dt)$  aus  $P'(t)$  und  $P''(t)$  berechnen.

Allerdings ergeben sich dabei verschiedene Nachteile. Das Hauptproblem ist eine Fehlerakkumulation während erfolgreicher Integration, was nach verschiedenen Zeitpunkten zu Spannungsinstabilitätsartefakten führt.

Eine bessere Lösung berücksichtigt den Zustand des Systems, der durch die Position  $P(t)$  und die Geschwindigkeit  $P'(t)$  dargestellt wird. Der Vektor  $Q(t)$  ist die Verknüpfung von  $P(t)$  mit  $P'(t)$  und der Vektor  $Q'(t)$  ist die Verknüpfung von  $P'(t)$  mit  $P''(t)$ .

Mit dieser Berechnung reduziert man das Problem auf eine Differentialgleichung erster Ordnung, für welche die Integrationstechniken existieren. Nun besteht die Aufgabe darin,  $Q(t + dt)$  aus  $Q(t)$  und  $Q'(t)$  zu berechnen.

## 2. Explizite Integrationsmethoden

Explizite Integrationsmethoden sind die einfachsten Methoden, die verfügbar sind, um solche normalen Differentialgleichungen erster Ordnung zu lösen. Sie berücksichtigen die Voraussage des Zustandes von dem zukünftigen System direkt aus dem Wert der Ableitungen. Die bekannteste explizite Technik sind die Runge-Kutta-Methoden. Unter ihnen berücksichtigt die Eulermethode den Zukunftszustand als eine direkte Extrapolation aus dem aktuellen Zustand und der Herleitung.

### 2.1. Methoden aus der Familie der Runge-Kutta

Der „intuitive“ Weg, das Inkrement einer Funktion aus seiner ersten Ableitung zu berechnen, ist, die Durchschnittsgeschwindigkeit der Funktion  $(Q(t + dt) - Q(t)) / dt$  gleich der berechneten Geschwindigkeit  $Q'(t)$  zu setzen:

$$Q(t + dt) = Q(t) + Q'(t)dt \quad (\text{F23})$$



Abb.22: Euler-Integration erster Ordnung

Den Simulationsfehler zu kontrollieren, ist jedoch in dieser Anwendung nicht nur für die Genauigkeit wichtig, sondern besonders für die Stabilität der Simulation. Um den Fehler zu reduzieren, kann man ein kleineres Zeitintervall  $dt$  nutzen, das proportional zur Berechnungszeit wächst.

Ein weiterer Weg, die Genauigkeit zu verbessern, wäre, die Ordnung der Simulation zu erhöhen, indem man Zwischenwerte berechnet, sodass sich die Fehler voriger Ordnungen vermindern.

In der Mittelpunktmethode zweiter Ordnung (Abb.23) ist der Simulationsfehler proportional zu  $dt^3$ . Dazu werden zwei Ableitungen zu den Zeiten  $t$  und  $t + dt/2$  benötigt. Das Endergebnis ist für eine Zwischenberechnung in der Mitte des Intervalls von Vorteil.

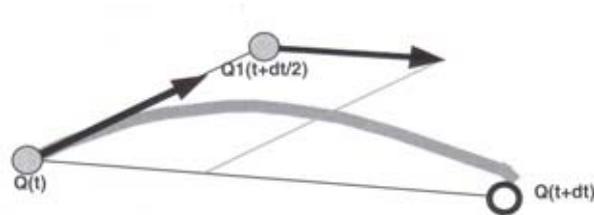


Abb.23: Mittelpunktmethode zweiter Ordnung

Die Runge-Kutta-Methode vierter Ordnung (Abb.24) hat einen Simulationsfehler proportional zu  $dt^5$ . Es werden die Zwischenwerte berechnet, der erste in der Mitte des Intervalls, wie in der Mittelpunktmethode, der zweite ebenfalls am Mittelpunkt, wobei er die Informationen des ersten benutzt, und der dritte am Ende des Intervalls, der wiederum die Information des zweiten benutzt. Das Endergebnis ist eine gewichtete Summe von Zwischenwerten am Ende des Intervalls. Dafür werden vier Ableitungen zu den Zeiten  $t$ ,  $t + dt/2$ ,  $t + dt/2$  und  $t + dt$  benötigt.

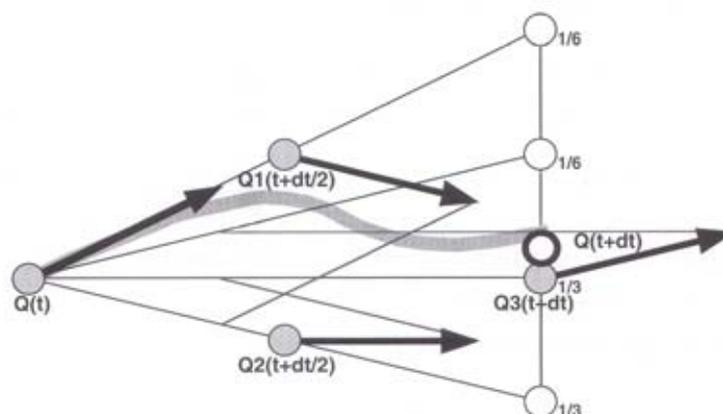


Abb.24: Runge-Kutta-Methode vierter Ordnung

In einer typischen Stoffsimulationsmethode hat sich letztere Methode als weit effizienter herausgestellt als die Mittelpunktmethode zweiter Ordnung, welche wiederum weitaus besser ist als die Eulermethode erster Ordnung. Obwohl die Runge-Kutta-Methode teurer ist, als die anderen beiden, lohnt sich deren Anwendung aufgrund des größeren Zeitintervalls, besonders wenn man die Vorteile wie z.B. den Gewinn an Stabilität und Genauigkeit berücksichtigt.

Um einen Vorteil aus dieser Durchführung zu ziehen, wird jedoch ein effizienter, adaptiver Kontrollalgorithmus benötigt, um das Zeitintervall für den Erhalt optimaler Werte einzustellen.

## 2.2. Zeitintervall, Genauigkeit und Ordnung der Integration

Runge-Kutta-Methoden sind Annäherungen der aktuellen Lösung und liefern ihre Ergebnisse innerhalb eines Fehlerbereichs, der von verschiedenen Faktoren abhängt: die Größe des Zeitintervalls, die Ordnung der Integration und die Regelmäßigkeit der Funktion, die das Differentialsystem bestimmt.

Die Runge-Kutta-Methoden erster Ordnung benutzen Polynom-Funktionen höherer Ordnung, um die Annäherung durchzuführen. Eine wachsende Ordnung bedeutet jedoch nicht wachsende Präzision. Die Situation ist analog zur Polynom-Interpolation, bei der zwar weniger Punkte benötigt werden, um eine Funktion mit einem Polynom hoher Ordnung zu interpolieren, dafür jedoch glatte Funktionen, die effektiv durch diese Polynome angenähert werden können. Bei hohen irregulären und diskontinuierlichen Funktionen sind diese Annäherungen nicht gültig. Ferner kann das Darstellen einer diskontinuierlichen Funktion, die regelmäßige Funktionen hoher Ordnung benutzt, zu katastrophalen Artefakten nahe der Unterbrechung führen. Simuliert man diese Funktionen mit Funktionen niedriger Ordnung oder sogar linearen Funktionen, die kleine Zeitintervalle benutzen, wächst die Robustheit enorm und reduziert die Fehler in solchen Situationen.

Methoden hoher Ordnung benötigen offensichtlich mehr Berechnungen pro Zeitintervall. Wie man aus obigem bereits erkennen kann, benötigt man für die Eulermethode erster Ordnung einen Integrationsschritt, zwei für die Mittelpunktmethode zweiter Ordnung und vier für die Runge-Kutta-Methode vierter Ordnung, wobei die Berechnungszeit proportional zu diesen Anforderungen ist.

Methoden hoher Ordnung kombinieren die Daten von verschiedenen Punkten, um die Ordnung der Genauigkeit von der Lösung zu erhöhen. Die Simulationsgenauigkeit verbessert sich mit der Ordnung der Methode, wenn das Zeitintervall kleiner wird. Das bedeutet, dass man für kleine Zeitintervalle eine sehr genaue Simulation erhält.

Die optimale Ordnung der Integration muss gemäß all dieser Überlegungen festgelegt werden. Die Runge-Kutta-Methode vierter Ordnung ist gewöhnlich ein sehr guter Kompromiss für die Anwendungen von Stoffsimulationen, wenn man ein normales elastisches Modell verwendet.

### 2.3. Fehlerauswertung

Die Fehlerauswertung ist eine gute Möglichkeit, die Angemessenheit des Zeitintervalls zu kontrollieren, das für die Berechnung verwendet wird.

Anstelle von vier werden sechs Ableitungsstufen verwendet, dadurch erhält man eine genaue Ordnung, sowie die Fehlerauswertung.

Obwohl dies 50% teurer ist, als die Runge-Kutta-Methode vierter Ordnung, ist diese Abschätzung dafür genauer. Dank der Fehlerauswertung kann das Zeitintervall optimal kontrolliert werden.

Der Berechnungsfehler kann auch dazu verwendet werden, den Effekt der Berechnungsfehler, die das Ergebnis ungewollter mechanischer Energieansammlung in der Struktur sind, zu reduzieren. Die Idee ist, den aktuellen Zustand nach dem Simulationsschritt auf eine Position innerhalb des Fehlerintervalls zu „korrigieren“.

Dies würde eine Positionskorrektur zur Minimierung der elastischen Deformationsenergie und eine Genauigkeitskorrektur zur Minimierung der deformierten kinetischen Energie erfordern.

### 2.4. Zeitintervall-Kontrolle

Das passendste Zeitintervall hängt von der Struktur des simulierten Problems (in diesem Fall: Topologie und Ruheposition des Federmassenmeshes), seinen mechanischen Eigenschaften und seinem aktuellen Zustand ab. Außerdem wechselt es den Verlauf der Simulation. Es gibt keine andere gute Methode zur Auswertung

des theoretisch optimalen Zeitintervalls, als den Simulationsfehler für jedes Zeitintervall zu überwachen (wie oben beschrieben).

Zu diesem Zweck kann eine numerische Technik, die auf einer Simulationsfehlervoraussage basiert, verwendet werden. Die Größe dieses Intervalls ist eine brauchbare und relevante Information, ob das auf ihm angewendete Zeitintervall für die Berechnung dieser Iteration gut angepasst wurde. Ein angemessener Wert dieses Fehlerintervalls kann Anpassungen im Zeitintervall durch einen Mechanismus, der auf Schwellwerten basiert, kontrollieren (Abb.25).

Der Schwellmechanismus verwendet drei benutzerdefinierte Konstanten  $E_m$ ,  $E_n$ ,  $E_p$  und versucht den aktuellen maximalen Simulationsfehler zwischen zwei Werten,  $E_m$  und  $E_p$ , zu halten.

- Ist der Fehler über  $E_p$ , wird das Zeitintervall durch eine gegebene Konstante dividiert. Der letzte Simulationsschritt wird mit diesem neuen Zeitintervall noch einmal berechnet.
- Ist der Fehler über  $E_n$ , wird das Zeitintervall durch eine gegebene Konstante dividiert.
- Ist der Fehler unter  $E_m$ , wird das Zeitintervall mit einer gegebenen Konstante multipliziert.

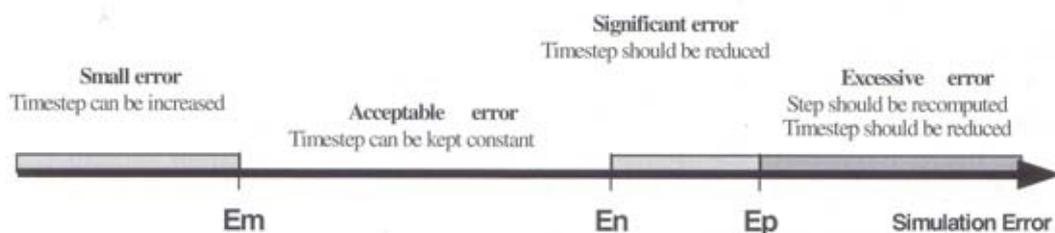


Abb.25: Simulationsfehler und Kontrolle des Zeitintervalls

Die Schwellwertkonstanten werden erfahrungsgemäß wie folgt gesetzt:

- $E_p$  wird auf einen Wert gesetzt, der unrealistische Ergebnisse und Instabilität verhindert.
- $E_n$  wird unter  $E_p$  gesetzt, um keine visuelle Ungenauigkeit zu erhalten.
- $E_m$  wird unter  $E_n$  gesetzt, um hohe Werte des Zeitintervalls zu gewährleisten.

### 3. Implizite Integrationsmethoden

Ungenauigkeit ist die einzige Überlegung bei der Bestimmung des angelegten Zeitintervalls. Eine weitere Streitfrage in der Partikelsimulation ist die numerische Steifheit. Bestimmte Integrationsmethoden versuchen, Integrationsungenauigkeit zu vermeiden, indem sie Simulationsinstabilität hervorrufen. Die Hauptidee ist, Rückwärtsintegrationsschritte (implizit) zu verwenden, die sich von gewöhnlichen Vorwärtsintegrationsschritten (explizit) unterscheiden. Sie finden den Wert zur Zeit  $t + dt$ , die den Wert zur Zeit  $t$  durch Anwendung Zeitentgegengesetzter Ableitungen produzieren würde.

Die grundlegendste Implementierung ist der Euler-Schritt, der die Berechnung der vorausgesagten Ableitung für das nächste Zeitintervall berücksichtigt. Der inverse Euler-Schritt (Abb.26) verwendet diese vorausgesagte Ableitung für die „Rückwärts“-Berechnung des Schrittes:

$$Q(t) = Q(t + dt) - Q'(t + dt)dt \quad (\text{F24})$$

Die Ableitung zur Zeit  $t + dt$  ist nicht bekannt und muss angenähert werden. Wegen der linearen Annäherung wird diese Integrationsmethode oft „semiimplizite Integrationsmethode“ genannt.

Das System ist stabilisiert, was auch immer für ein Zeitintervall verwendet wird.

Die Stabilisierung erhält man, indem man das lineare Gleichungssystem, das die Entwicklung der Ableitungen in Beziehung bringt, löst. Letztendlich wird hierfür die inverse Matrix, die in Gleichung (F27) benötigt wird, ausgeführt.

Die Struktur der Matrix sollte der Beziehungsstruktur zwischen den Partikeln ähnlich sein. Im Zusammenhang mit der Partikelsimulation interagieren die Partikel nur mit einer konstanten Anzahl von Nachbarn, stattdessen ist die Matrix schwach besetzt. Somit sollte folgendes lineare Gleichungssystem gelöst werden:

$$\left( I - \frac{\partial Q'}{\partial Q}(t)dt \right) (Q(t + dt) - Q(t)) = Q'(t)dt \quad (\text{F25})$$

Alle impliziten Methoden sind stabiler, aber nicht notwendigerweise genauer als ihre expliziten Gegenstücke. Sie liefern nur aufgrund der Voraussetzung, dass ein Ergebnis, nahe am Gleichgewicht besser ist als eines das nur durch Extrapolation gewonnen wird, bessere Ergebnisse.

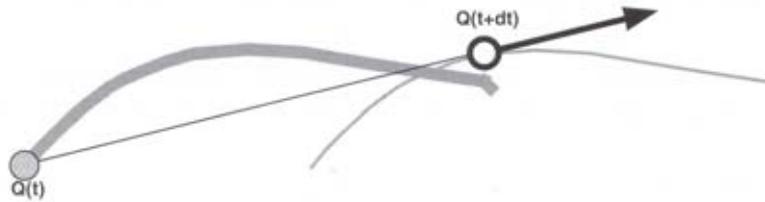


Abb.26: Der inverse Euler-Schritt

#### 4. Erhaltung der Simulationsstabilität

Die numerische Integration, sowie jeder andere numerische Prozess, ist ungenau. Eine kleine und kontrollierte Ungenauigkeit wirkt sich nicht sehr stark auf das Ergebnis aus, allerdings kann numerische Ungenauigkeit einen ernsteren Seiteneffekt hervorrufen: die Simulation könnte instabil werden.

Meistens muss in einer Simulation viel mehr die Instabilität als die numerische Ungenauigkeit kontrolliert werden. Der Hauptgrund, das Augenmerk auf die Simulationsgenauigkeit zu legen, ist nicht der visuelle Realismus, sondern der Schutz der Simulation vor einer „Explosion“. Während kleine, unrealistische Artefakte oft unbemerkt in die Echtzeit-Simulation eingehen, führt eine numerische Explosion zu unauffindbaren Effekten.

##### 4.1. Stabilität, Simulationszeitintervall und innere Schwingungen

Ein Partikelsystem, das aus Massen und Federn besteht, ist nicht äquivalent zu einer elastischen Oberfläche. Jedes Partikel kann individuell schwingen und ist von dem Effekt seiner angrenzenden Federn abhängig.

Eine genaue mechanische Simulation sollte in der Lage sein, die verschiedenen Schwingungsarten genau zu simulieren. Schwingungen eines einzelnen Partikels sind für die Simulation nicht besonders relevant, da sie nur das Ergebnis der Feder-Massen-Diskretisierung sind. Ideal wäre natürlich, wenn sie in der Simulation gar nicht erst vorkommen würden.

Zur Simulation einer genauen Schwingungsart sollte sich das Simulationszeitintervall auf die Schwingungsfrequenz beziehen. Somit kann das Schwingungsmuster genau mithilfe der Zustände der Simulation beschrieben werden. Für eine genaue globale Simulation sollte das Zeitintervall an die maximale Frequenz aller möglichen

Schwingungsarten des Systems angepasst werden. Je höher diese Frequenz ist, desto kleiner sollte das Zeitintervall sein.

In einem Gitter mit Partikeln der Masse  $M$ , die durch Federn der Länge  $l$  und den metrischen Elastizitätskoeffizient  $k$  verbunden sind, ist die lokale Schwingungsfrequenz proportional zu  $(k/(Ml))^{-1/2}$ . Dieses System modelliert eine elastische Oberfläche mit einer Oberflächendichte von  $M/l^2$  und einem Young-Betrag für die Oberfläche von  $k/l$ . Die Skizze verdeutlicht, dass die gleiche Oberfläche mit halb so großen Elementen Partikel mit einer Masse von  $M/4$  benötigt, die durch Federn der Länge  $l/2$  und den metrischen Koeffizienten  $k/2$  verbunden sind. Die lokale Schwingungsfrequenz dieser Partikel verdoppelt sich gegenüber dem vorherigen Modell.

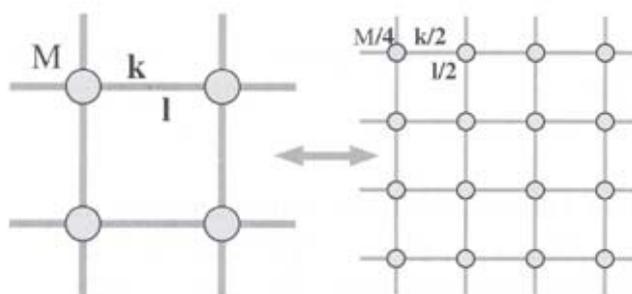


Abb.27: Eine metrische Oberflächenelastizität mit zwei verschiedenen Feder-Massen-Systemen

Das erlaubte Simulationszeitintervall einer Partikelsimulation ist für Oberflächen mit äquivalenten mechanischen Parametern proportional zur Größe des Elementes  $l$ . Des weiteren wird die totale Simulationszeit proportional zu  $1/l^3$ , wenn die Anzahl der Elemente proportional zu  $1/l^2$  ist. Aus diesem Grund benötigt eine schnelle Simulation von Partikelsystemen Diskretisierungen.

#### 4.2. Implizite vs. explizite Integrationsmethoden

Implizite Methoden haben im Gegensatz zu expliziten Methoden keine Probleme mit der Instabilität. Zur Verdeutlichung soll folgende lineare Differentialgleichung dienen:

$$X(t) + kX'(t) = 0 \quad (\text{F26})$$

Die Lösung beginnt bei dem Initialisierungswert  $X(0)$  und konvergiert mit der Zeit gegen null. Die Konvergenzrate steigt mit dem Wert des Parameters  $k$  (Abb.28).

Die Euler-Simulation dieser Gleichung ist eine Folge von Werten  $X_n$ . Sie werden für jeden Zeitpunkt  $t_n = ndt$  berechnet und nutzen die Gleichung (F23) folgendermaßen:

$$X_0 = X(0)$$

$$X_{n+1} = X_n + X'_n dt = X_n \frac{k - dt}{k} \quad (\text{F27})$$

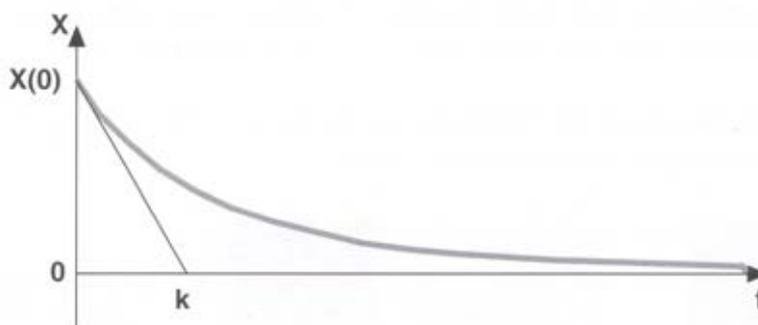


Abb.28: Lösung der Differentialgleichung

Diese Folge von Werten legt verschiedene Verhaltensarten dar, die von den relativen Werten des Zeitintervalls  $dt$  und  $k$  abhängen. Dies sind die Fälle, die auftauchen können:

- Wenn  $dt$  kleiner ist als  $k$ , konvergiert die Folge fortschreitend gegen 0. Je kleiner  $dt$  ist, desto näher liegen die Werte der Folge an denen der Exponentialkurve.
- Wenn  $dt$  zwischen  $k$  und  $2k$  liegt, konvergiert die Folge gegen 0, indem sie abwechselnd positive und negative Werte annimmt. Das bedeutet, dass hier Schwingungen auftreten. Je größer  $dt$  ist, desto langsamer konvergiert die Folge.
- Wenn  $dt$  größer als  $2k$  ist, divergiert die Folge und die Werte werden größer und größer. Dies nennt man numerische Instabilität.

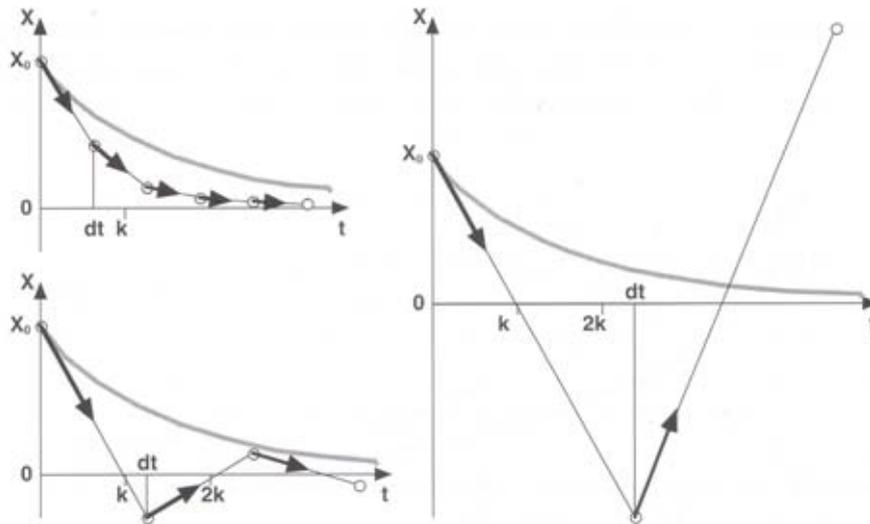


Abb.29: Lösung der Gleichung (F31) mithilfe des Euler-Schrittes

Die Skizzen zeigen deutlich das mögliche Verhalten der Integrationsmethoden, die von der Zeitintervallgröße  $dt$  relativ zur Zeitkonstanten  $k$  abhängen. Numerische Instabilität tritt also auf, wenn das Zeitintervall zu groß wird.

Bei einer genauen Simulation sollte das Zeitintervall mit der Zeitkonstanten des Systems vergleichbar bleiben. In großen Systemen kann schon eine kleine, lokale Zeitkonstante in nur einem Teil des Systems zur Destabilisierung des gesamten Systems führen.

Wie bereits oben erwähnt, sind die impliziten Methoden für Probleme mit Instabilität von besonderer Bedeutung.

Sollte  $dt$  verglichen mit  $k$  relativ groß werden, konvergiert die Folge schnell gegen 0 (Abb.30).

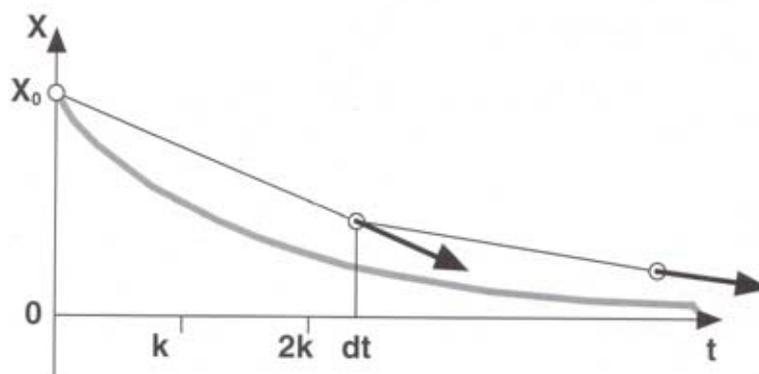


Abb.30: Konvergenzverhalten des Euler-Rückwärtsschrittes mit großen Zeitintervallen

Dieses Beispiel verdeutlicht die Überlegenheit der impliziten Methoden in Hinblick auf die numerische Stabilität. Der Gleichgewichtszustand des Systems, gegen den die Simulation konvergieren soll, lässt sich mithilfe einer Abschätzung der hergeleiteten Entwicklung finden.

Der einzige, aber enorme Nachteil der impliziten Methoden ist, dass die Linearsystemgleichungen Probleme mit großen Zustandsvektoren mit sich bringen.

### 2.5.2. Auswahl der geeigneten Integrationsmethode

Für ein gegebenes Problem sollte die passende Integrationsmethode definiert werden, wobei man sich über folgende Punkte klar werden sollte:

- Die benötigte Genauigkeit der Simulation, die das Zeitintervall begrenzt
- Die benötigte Genauigkeit für ein gegebenes Zeitintervall, das mit der Genauigkeitsordnung zusammenhängt
- Die numerische Stabilitätsfrage, wodurch ebenfalls das Zeitintervall begrenzt wird
- Andere Faktoren, die das Zeitintervall begrenzen, wie beispielsweise Hochfrequente Phänomene, die rekonstruiert werden müssen, oder eine genaue Kollisionserkennung und –antwort
- Die Summe der Zeit, die für die Berechnung eines Zeitintervalls benötigt wird
- Die Struktur des Problems, das die Vereinfachung bestimmter Berechnungsaufgaben erlaubt.

Offensichtlich liegen die Vorteile der impliziten Methoden in den meisten Anwendungen der Computergrafik, in denen die numerische Stabilität das Hauptkriterium ist.

Die meisten Partikelsysteme, die zur Stoffsimulation verwendet werden, sind Steifheitssysteme. Dabei ist das wichtigste Verhalten die globale Stoffbewegung. Explizite Methoden benötigen Zeitintervalle, die an die Frequenzen dieser Schwingungen angepasst werden. Dadurch wird versucht, numerische Instabilität zu verhindern. Anhand expliziter Methoden lässt sich die Bewegung jedes Partikels genau berechnen.

Abhängig von der Art der Anwendung, die das mechanische System integriert, können verschiedene Möglichkeiten in Betracht gezogen werden:

- Für Echtzeit- und interaktive Systeme sind einfache Partikelsysteme mit schneller und annähernder Integrationsmethode sehr geeignet. Wenn das System einfach genug ist, Vereinfachungen, die die Berechnungszeit jedes Zeitintervalls für die korrekte Framerate verringern, zu implementieren, können implizite Methoden gewählt werden. Dafür verwendet man reguläre Meshes und einfache, lineare Interaktionen zwischen den Partikeln. Explizite Methoden machen nur bei komplexen Modellen oder bei schnellen Iterationen auf groben Meshes Sinn.
- Bei visuellen Offline-Simulationssysteme können implizite Methoden für steife Systeme verwendet werden, allerdings auf Kosten der Genauigkeit. Explizite Methoden hoher Ordnung sind sinnvoll, da sie eine höhere Genauigkeit für kleine Zeitintervalle zur Verfügung stellen.
- Präzise Simulationssysteme verwenden meist komplexe mechanische Modelle auf der Grundlage von verfeinerten Meshes. Auch stetige mechanische Modelle und Partikelsysteme mit expliziten Integrationsmethoden hoher Ordnung wären hierbei möglich.

Die richtige Wahl des Systems hängt von dem jeweiligen Kontext und der Situation ab. Die allgemeine Regel ist, implizite Methoden zur robusten Simulation steifer Systeme mit großen Zeitintervallen, und explizite Methoden zur genauen Simulation komplexerer Systeme und Verhalten heranzuziehen.

### 3. Realisierung eines Partikelsystems

Ein Partikelsystem zu realisieren erfordert viele verschiedene Überlegungen und Berücksichtigungen, das geht bereits aus den ersten beiden Kapiteln hervor. Zunächst musste ich mich entscheiden, ob ich ein Mesh aus Dreiecken oder aus Quadraten verwende. Danach hab ich mir Schritt für Schritt ein Partikelsystem aus den Komponenten `Particle`, `Spring` und `Vector` zusammengesetzt. Auf jedes Partikel wirken verschiedene Kräfte, die in einer Methode `updateMesh()` berechnet und in der Methode `StepSimulation()` addiert werden. Gezeichnet wird das Mesh in der Methode `display()`.

Programmiert habe ich in der Sprache C++ mit Hilfe von OpenGL für die graphische Anbindung.

#### 3.1. Softwaretechnischer Entwurf

##### 3.1.1. Aufbau des Partikelsystems mit seinen Klassen

Die Partikelmodellierung von Stoffen basiert auf einem Netzwerk von Partikelknoten bzw. Partikeln (Abb.31).

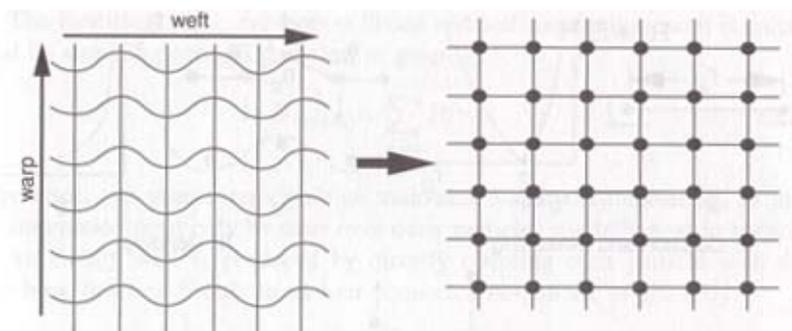


Abb.31: Netzwerk von Partikeln

Ich habe mich für ein Mesh aus Quadraten entschieden, da es einfacher zu berechnen und meiner Meinung nach übersichtlicher ist. Das Mesh wird folgendermaßen aufgebaut:

Jedes Quadrat wird im Uhrzeigersinn gezeichnet, beginnend mit der vorderen Ecke. Danach wird das daneben liegende in z-Richtung gezeichnet usw. Wenn diese Reihe fertig ist, wird vorne das nächste Quadrat in x-Richtung gezeichnet, dann wieder alle daneben liegenden in z-Richtung, bis das Mesh vollständig gezeichnet ist. Abb. 32



Der Konstruktor und der Default-Konstruktor dazu sehen folgendermaßen aus:

```
Particle ( float m, float invm, Vector &p, Vector &v, Vector &a,
          Vector &f, Vector &d, Vector &dr, Vector &w, bool fx );
Particle ();
```

Das Mesh mit den Partikeln wird als zweidimensionales Array im Konstruktor der Klasse `ParticleSystem` generiert:

```
mMesh = new Particle*[iResWidth];      //Anzahl der Particle
for (int i=0; i<iResWidth; i++)
{
    mMesh[i] = new Particle[iResLength];
}
```

Wie groß man das Mesh maximal und minimal wählen sollte, werde ich zu einem späteren Zeitpunkt erläutern.

### 3.1.1.2. Die Klasse `Spring`

Jedes Partikel ist über verschiedene Federn miteinander verbunden. Die Federn sind allerdings nicht zu sehen, da sie nicht explizit in der `display`-Methode gezeichnet werden, sondern nur imaginär vorhanden sind. Sie halten die Partikel zusammen und wirken Belastungen entgegen. Es gibt zwei Arten von linearen Federn in meinem Partikelsystem, die die elastischen Eigenschaften des Stoffes simulieren:

- Die horizontalen und vertikalen Federn bewahren die Grundstruktur des Meshes,
- die diagonalen Federn widersetzen sich den Scherkräften und verleihen dem Stoff eine gewisse Stärke. Ohne diese Scherfedern wäre der Stoff ziemlich elastisch.

In Abb.33 sind die verschiedenen Arten von Federn skizziert, wobei die rot-gezeichneten Krümmungsfedern in meinem Mesh nicht auftreten.

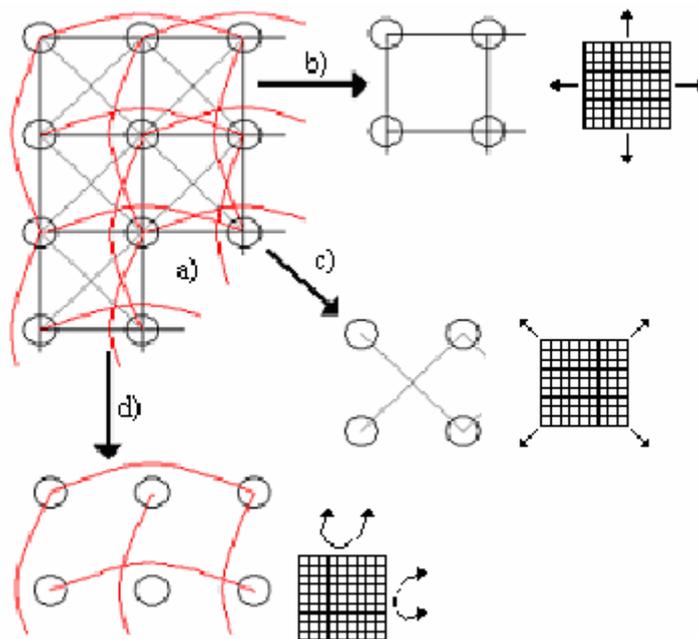


Abb.33: Die Feder-Massen-Topologie: a) globale Struktur, b) vertikale und horizontale Federn, c) Scherfedern oder diagonale Federn, d) Krümmungsfedern

Die Klasse `Spring` enthält alle Eigenschaften einer Feder. `mBegin` zeigt auf das erste Partikel, ist somit der Anfangspunkt der Feder, `mEnd` ist der Endpunkt einer Feder und zeigt demnach auf das zweite Partikel. Die Ruhelänge wird explizit für jede einzelne Feder berechnet. Sind der erste und der zweite Punkt der Feder ungleich null, wird die Differenz der Positionen dieser beiden Partikel und von diesem resultierenden Vektor die Länge berechnet. Ansonsten wird eine Ruhelänge von 1.0 angenommen:

```
if (first != 0 && second != 0)
{
    mLength = (first->getPosition() - second->getPosition()).
                computeLength();
}
else
    mLength = 1.0;
```

Die Parameter Ruhelänge, Federkonstante und Dämpfung werden zur Berechnung der Federkräfte zwischen jedem verbundenen Partikelpaar benötigt.

Der Default-Konstruktor und der Konstruktor

```
Spring ();
Spring( Particle* first, Particle* second, float sc, float d );
```

ergeben sich aus folgender Klasse:

```
class Spring
{
    private:
        Particle* mBegin;    // Zeiger auf erstes Particle
                           // (Anfangspunkt der Feder)
        Particle* mEnd;     // Zeiger auf zweites Particle
                           // (Endpunkt der Feder)

        float mLength;     // Ruhelänge der Feder
        float fSpringConst; // Federkonstante
        float fDamping;     // Dämpfung
}

```

Die Federn werden ebenfalls in einem Array im Konstruktor der Klasse

ParticleSystem generiert, allerdings ist dieses Array nur eindimensional:

```
mSprings = new Spring[iSpringCount];    //Anzahl der Springs

//Schleife über alle Particle
int springIndex = 0;
for ( b=0; b<iResWidth; b++)
{
    for ( l=0; l<iResLength; l++)
    {
        /* Initialisierung der Springs mit Adresse(&) der Particles */

        //horizontale Federn (parallel zur x-Achse)
        if (b < iResWidth-1)
        {
            mSprings[springIndex] =
                Spring( &mMesh[b][l], &mMesh[b+1][l] ,
                    SPRINGTENSIONCONSTANT, SPRINGDAMPINGCONSTANT );
            springIndex++;
        }

        //vertikale Federn (parallel zur z-Achse)
        if (l < iResLength-1)
        {
            mSprings[springIndex] =
                Spring( &mMesh[b][l], &mMesh[b][l+1] ,
                    SPRINGTENSIONCONSTANT, SPRINGDAMPINGCONSTANT );
            springIndex++;
        }

        //diagonale Federn (von rechts unten nach links oben)
        if ((b < iResWidth-1) && (l < iResLength-1))
        {
            mSprings[springIndex] =
                Spring( &mMesh[b][l], &mMesh[b+1][l+1] ,

```

```

        SPRINGSHEARCONSTANT, SPRINGDAMPINGCONSTANT );
springIndex++;
}

//diagonale Federn (von rechts oben nach links unten)
if ((b < iResWidth-1) && (l > 0))
{
mSprings[springIndex] =
    Spring( &mMesh[b][l], &mMesh[b+1][l-1] ,
        SPRINGSHEARCONSTANT, SPRINGDAMPINGCONSTANT );
springIndex++;
}
}
}

```

Alle Federn werden mit Hilfe des `springIndex` durchnummeriert. Anfangs wird dieser Index auf null gesetzt, sobald eine Feder – imaginär – gezeichnet wurde, wird er um eins erhöht. Die Anzahl der Federn steht im Konstruktor des `ParticleSystem` und kann mit Hilfe einer Formel berechnet werden.

### 3.1.1.3. Die Klasse `Vector`

Da die Parameter der Klasse `Particle` fast alle vom Typ `Vector` sind, habe ich mir noch eine Klasse `Vektor` mit drei Komponenten, `x`, `y` und `z`, und verschiedenen Methoden und Operatoren definiert, die die grundlegenden Vektoroperationen implementieren.

```

class Vector
{
    float x;
    float y;
    float z;
    float tol;
}

```

Die acht Operatoren sind : Addition ( + ), Subtraktion ( - ), Multiplikation ( \* ), Division ( / ), Skalar-Multiplikation ( \*= ), Skalar-Division ( /= ), Zuweisung ( = ) und Skalarprodukt ( \* ).

Die Klasse hat außer einem Konstruktor und einem Default-Konstruktor noch zusätzlich einen Copy-Konstruktor:

```

Vector ( float x, float y, float z );

```

```
Vector ();
Vector ( const Vector& in );
```

### 3.1.1.4. Die Klasse `ParticleSystem`

Die Größe des Meshes wird in der Klasse `ParticleSystem` festgelegt.

Eigenschaften wie die Breite und die Länge, sowie die Auflösung der Breite und der Länge, als auch die Anzahl der Federn sind alle vom Typ `int`. Der letzte Parameter, die Höhe des gesamten Meshes, ist vom Typ `float`. Die Angabe über die beiden Arrays – ein zweidimensionales von Partikeln und ein eindimensionales von Federn – wird außerdem in dieser Klasse generiert.

```
class ParticleSystem
{
    public:
        int    iWidth, iLength;           // Breite, Länge des
                                           Meshs
        int    iResWidth, iResLength;    // Auflösung Breite,
                                           Länge des Meshs
        int    iSpringCount;             // Anzahl der Federn
        float  fHeight;                  // Höhe des gesamten
                                           Meshs

        Particle** mMesh;                // 2D-Array von Particles
        Spring* mSprings;                 // 1D-Array von Springs

        ...
}
```

Der Konstruktor und Default-Konstruktor ergeben sich wie folgt:

```
ParticleSystem (int b, int l, int rb, int rl, int c, float h);
ParticleSystem ();
```

### 3.1.2. Wichtige Methoden

In der `main`-Funktion wird zunächst auf dem Objekt `test` in `ParticleSystem` die Methode `GenerateMesh()` aufgerufen. Die danach folgenden fünf Routinen aus Open GL sind zur Initialisierung des Fensters notwendig:

```
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
glutInitWindowSize (700, 700);
glutInitWindowPosition (100, 100);
```

```
glutCreateWindow ("Christiane's Partikelparty");
```

Als nächstes wird die `init`-Methode, daraufhin die `display`-Methode, anschließend die `keyboard`-Funktion und zuletzt die `MainLoop` durchgeführt.

Damit sich die folgenden Funktionen auf jedes Partikel beziehen, stehen am Anfang jeder Methode zwei `for`-Schleifen. Die äußere Schleife läuft über jedes `b`, also die Breite, die innere über jedes `l`, die Länge des Meshes.

```
for ( int b=0; b<iResWidth; b++ )
{
    for ( int l=0; l<iResLength; l++ )
    {
        ... /* Anweisungen */ ;
    }
}
```

Im Initialisierungsteil wird den Schleifenvariablen `b` und `l` der Anfangswert `0` zugewiesen. Solange die Bedingung erfüllt ist, dass die beiden Schleifenvariablen kleiner als die jeweilige Auflösung sind, wird die Schleife iteriert. In der Modifikation wird festgelegt, dass der Wert der Schleifenvariablen von Iteration zu Iteration um `1` erhöht wird. Danach folgen verschiedene Anweisungen, die von Methode zu Methode variieren.

Um jede Feder in eine Methode einzubeziehen, genügt eine einzige `for`-Schleife, da dieses Array nur eindimensional ist. Sie läuft über jedes `i`, also die Anzahl der Federn. Die Schleife sieht ähnlich aus wie die obigen beiden, mit dem Unterschied, dass hier die Schleife abgebrochen wird, wenn die Bedingung, `i` muss kleiner als die Anzahl der Federn, `iSpringCount`, sein nicht mehr erfüllt ist.

### 3.1.2.1. Die Methode `GenerateMesh()`

Zur Initialisierung des gesamten Meshes mit all seinen verschiedenen Werten dient die Methode `GenerateMesh()`. Sie füllt die Datenstrukturen für die Partikel und Federn. Zunächst wird jedem Partikel eine Masse zugewiesen. Berechnet wird sie, indem man die Massen einer Fläche, die das Partikel teilt, summiert und das ganze mit einem Drittel multipliziert:

```
(( f * MASSPERFACE ) / 3) .
```

Je nachdem wo ein Partikel liegt, ergibt sich ein anderer `float`-Wert `f` für die Berechnung der Masse:

- Der erste und der letzte Punkt des Meshes haben den Wert eins,
- die anderen beiden Eckpunkte den Wert zwei,
- alle Punkte am rechten und linken Rand den Wert drei
- und die restlichen Punkte den Wert sechs.

Die Masse pro Fläche berechnet sich aus der Gesamtmasse der Kleidung dividiert durch das Produkt der Breite und Länge des Meshes mal zwei.

Als nächstes werden die Partikel auf eine bestimmte Position gesetzt und alle weiteren Parameter aus der Klasse `Particle` initialisiert. Am Ende der Methode werden noch die Partikel festgelegt, die durch eine Kraft bewegt werden können und die fest sind.

### 3.1.2.2. Die Methode `UpdateMesh()`

Die Methode `UpdateMesh()` wird über die `keyboard`-Funktion aufgerufen, indem die Taste „s“ – für „Simulation“ – gedrückt wird. Alle Kräfte, die auf das Mesh ausgeübt werden können, werden in der Methode `UpdateMesh()` berechnet und sofort mit Hilfe einer Methode `addForces()` aufaddiert. Die Schwerkraft, der Reibungsluftwiderstand und der Wind werden abhängig von jedem Partikel berechnet, die Feder-Dämpfungs-Kraft wird daraufhin abhängig von jeder Feder berechnet. Wie die Formeln dieser Kräfte genau aussehen, werde ich in dem Kapitel über Kräfte genauer erläutern.

### 3.1.2.3. Die Methode `display()` und `init()`

In der `display`-Methode werden alle Polygone gezeichnet, die dann zusammengefügt das Mesh ergeben. Die Reihenfolge des Zeichnens habe ich oben bereits erläutert. Des weiteren werden hier die Farbe des zu zeichnenden Objekts, die Einheitsmatrix und die „View Transformation“ angegeben.

Die `init`-Methode initialisiert GLUT, das OpenGL Utility Toolkit, und verarbeitet jedes Kommandozeilenargument, wie die Hintergrundfarbe des Fensters, den Viewport, die Perspektive und die Projektionen.



### 3.3. Simulationsalgorithmus

#### 3.3.1. Kräfte

Eine Kraft ist dafür verantwortlich, dass sich ein Objekt bewegt oder präziser gesagt, sie ändert die Beschleunigung eines Objekts. Egal, welche Kraft auf ein Partikel ausgeübt wird, an jedem Knotenpunkt wird ein Gleichgewicht an Kräften erzwungen. Dies ist auf das 3. Gesetz von Newton zurückzuführen, welches besagt, dass für jede Kraft eine gleiche Gegenkraft existiert: „keine Aktio ohne Reaktio“. Das bedeutet, dass eine Einzelkraft nicht für sich alleine existieren kann.

Zunächst werden die Kräfte in der Methode `GenerateMesh()` auf Null gesetzt. Danach werden sie in der Methode `UpdateMesh()` aufgerufen, berechnet und sofort aufaddiert.

##### 3.3.1.1. Schwerkraft

Die Schwerkraft ist eine konstante, nach unten wirkende Kraft auf ein Partikel proportional zu seiner Masse. Dabei ist seine Beschleunigung konstant  $-9.81 \text{ m/s}^2$ :

```
mMesh[b][l].setForces
    ( Vector ( 0,
              (float) (GRAVITY * (mMesh[b][l].getMass())),
              0 ));
```

Die Schwerkraft ist also das Produkt aus Masse und Gravitationsfaktor. Als entgegenwirkende Kraft dient in einem meiner Ergebnisse die des Windes. Im Falle der Tischdecke kann man die fixierten Punkte als die entgegenwirkende Kraft interpretieren, obwohl sie nicht extra in die Berechnung mit eingeht. Genauso verhält es sich mit dem fallenden Tuch, bei dem ein Punkt fixiert ist.

##### 3.3.1.2. Reibungsluftwiderstand

Der Reibungsluftwiderstand wirkt der Bewegung der Partikel entgegen. In meinen Ergebnissen wird man später sehen, dass die Simulation mit Reibungsluftwiderstand weitaus realistischer aussieht, als ohne.

Zunächst wird ein Widerstandsvektor namens `Drag` berechnet, der neben dem Widerstandskoeffizienten der Geschwindigkeit der jeweiligen Partikel entgegenwirkt. Das passiert mit Hilfe der Methode `Reverse()`. Danach wird er normalisiert.

```
Vector Drag = (mMesh[b][l].getVelocity());
Drag.Reverse();
Drag.Normalize();
mMesh[b][l].setDrag (Drag);
mMesh[b][l].addForces
    ( mMesh[b][l].getForces ()
    + mMesh[b][l].getDrag ()
    * ( mMesh[b][l].getVelocity().computeLength ()
    *   mMesh[b][l].getVelocity().computeLength () )
    * DRAGCOEFFICIENT );
```

Der Reibungsluftwiderstand lässt sich berechnen, indem man die Geschwindigkeit des Partikels quadriert und mit dem negativen Widerstandskoeffizienten multipliziert. Das Minus-Zeichen bedeutet hier, dass diese Kraft der Bewegung entgegenwirkt. Bei sich langsam bewegenden Objekten wird die Geschwindigkeit übrigens nicht quadriert, nur bei sich schnell bewegenden. Schnell bewegend impliziert, dass die Strömung um das Objekt herum nicht laminar<sup>6</sup>, sondern turbulent ist. Diese Gleichung ist sehr vereinfacht und reicht nicht für die praktische Analyse von Strömungsproblemen. Für die Simulation von Computerspielen jedoch bieten sie verschiedene Vorteile. Am offensichtlichsten ist, dass sie leicht zu implementieren sind; man muss nur die Geschwindigkeit des Objektes und einen angenommenen Wert für den Widerstandskoeffizienten wissen.

### 3.3.1.3. Windkraft

Für die Windkraft wird bereits in der Methode `GenerateMesh()` der Vektor `Wind` initialisiert. Hier wird die Windrichtung angegeben, somit bleibt entweder der `x`- oder der `z`-Wert Null, da es sonst eher unrealistisch aussehen würde.

```
mMesh[b][l].setWind
    (Vector (WindRandom(0, 10), WindRandom(0, 5), 0));
```

Ich habe in die Berechnung der Windkraft noch einen Zufälligkeitwert für die Stärke des Windes, `WindRandom`, miteinbezogen. Anhand dessen wird dafür gesorgt, dass

---

<sup>6</sup> laminare Strömung = eine Strömung, deren Verhalten durch innere Reibung bestimmt wird

der Stoff weit genug aus der vertikalen Ebene heraus beunruhigt wird, wodurch der Stoff viel realistischer flattert.

```
int WindRandom (int min, int max)
{
    int number;

    number = (((abs(rand()))%(max-min+1))+min));

    if (number>max)
    {
        number = max;
    }

    if (number<min)
    {
        number = min;
    }

    return number;
}
```

Daraufhin wird in der Methode `UpdateMesh()` der Vektor `Wind` aus den gegebenen Werten berechnet und dann normalisiert.

```
Vector Wind = (mMesh[b][l].getWind());
Wind.Normalize();
mMesh[b][l].addForces
    ( mMesh[b][l].getForces()
    + mMesh[b][l].getWind()
    * WindRandom(0, WindForceFactor) );
```

Die Windkraft ist das Produkt aus dem Vektor `Wind` und dem Zufälligkeitswert des Windes. Dieser Wert kann dann zwischen gar keinem Wind (0) und dem `WindForceFactor`, der sich beliebig ändern lässt, variieren.

#### 3.3.1.4. Feder-Dämpfungskraft

Die wichtigste Kraft, ohne die sich das Mesh gar nicht realistisch bewegen lassen würde und die dafür sorgt, dass die Partikel nicht zu weit auseinander gezogen werden, ist die Feder-Dämpfungskraft. Hierbei geht die `for`-Schleife nur über alle Federn – im Gegensatz zu allen vorherigen Kräften. Da alle Federn bereits initialisiert wurden, ist es einfach, die Daten für jede Feder herauszuholen und die Feder-Dämpfungskraft auf die verbundenen Partikel anzuwenden.

```

Vector f1, f2;          // Kraft von first und Gegenkraft von second

first->setDisplacement
    ( Vector ( first->getPosition() - second->getPosition() ) );
Vector Velo = first->getVelocity() - second->getVelocity();

float Springforce      = ((first->getDisplacement().computeLength()
    - mSprings[i].getLength())
    * mSprings[i].getSpringConst());

float Dampingforce     = (((Velo * first->getDisplacement())
    / first->getDisplacement().computeLength())
    * mSprings[i].getDamping());

float SpringDamping = -(Springforce + Dampingforce);

f1 = ( first->getDisplacement()
    / first->getDisplacement().computeLength() )
    * SpringDamping ;

f2 = f1;
f2.Reverse();

if ( first->bFixed == false )
    first->addForces ( f1 );

if ( second->bFixed == false )
    second->addForces ( f2 );

```

Die Federkraft folgt dem Gesetz von Hook und ist eine Funktion von gedehnter oder komprimierter Länge  $L$  der Feder relativ zu deren Ruhelänge  $r$  und deren Federkonstante  $sc$ . Die Federkonstante ist eine Größe, die die Kraft, ausgeübt durch die Feder, auf ihre Auslenkung bezieht:

$$F_s = sc(L - r) \quad (\text{F28})$$

Wenn die Feder mit zwei Objekten verbunden ist, wirkt sich  $+F_s$  auf das eine und  $-F_s$  auf das andere aus; dies sind die gleiche und die Gegenkraft.

Die Dämpfung wird für gewöhnlich in Verbindung mit Federn in numerischen Simulationen verwendet. Sie verhalten sich wie der Reibungsluftwiderstand, in denen die Dämpfung der Geschwindigkeit entgegenwirkt. In diesem Fall, wenn die Dämpfung zwischen zwei Partikeln verbunden ist, die sich aufeinander zu oder voneinander weg bewegen, sind die Dämpfer dafür zuständig, die relative Geschwindigkeit zwischen den beiden Partikeln zu verringern. Die Dämpfungskraft ist proportional zu der relativen Geschwindigkeit der verbundenen Partikel,  $v_1$  und

$v_2$ , und einer Dämpfungskonstante  $d$ , die relative Geschwindigkeit auf die Dämpfungskraft bezieht:

$$F_d = d(v_1 - v_2) \quad (\text{F29})$$

Federn und Dämpfung werden zu einem Feder-Dämpfungs-Element zusammengefügt, woraus sich die Formel ergibt, die ich auch implementiert habe:

$$\vec{F}_1 = -\{sc(L-r) + d[(\vec{v}_1 - \vec{v}_2) * \vec{L}] / L\} \vec{L} / L \quad (\text{F30})$$

Dies ist die Kraft die auf das erste Partikel ausgeübt wird, also auf `first`, auf `second` wird die Kraft  $\vec{F}_2 = -\vec{F}_1$  ausgeübt.  $\vec{L}$  ist die Länge der Feder-Dämpfung – also die Vektordifferenz der Positionen von `first` und `second` – und  $L$  ist die Größe des Vektors  $\vec{L}$ .

Federn und Dämpfung sind sehr nützlich bei der Simulation von Partikeln. Die Federkraft stellt die Struktur, welche die beiden Partikel zusammenhält zur Verfügung. Die Dämpfung gleicht die Bewegung zwischen den verbundenen Partikeln aus, damit sie nicht zu ruckartig oder zu elastisch wird. Des weiteren ist die Dämpfung für die numerische Stabilität von Bedeutung, damit die Simulation nicht explodiert.

### 3.3.2. Richtige Wahl der verschiedenen Konstanten

Alle wichtigen Parameter und Konstanten habe ich in einer Menge von globalen Definitionen in der Header-Datei von `ParticleSystem` platziert. Dort kann ich sie jederzeit problemlos ändern, ohne erst den ganzen Quellcode durchsuchen zu müssen. Für ein Mesh mit diesen Werten

```
IWIDTH          7
ILENGTH         7
IRESWIDTH       70
IRESLENGTH     70
```

haben sich folgende Parameter als am robustesten herausgestellt:

```
GRAVITY          -9.81      // Einheit: m/s^2
SPRINGTENSIONCONSTANT 400.0f    // Federkonstante
                                     vertikal & horizontal
SPRINGSHEARCONSTANT 600.0f    // Federkonstante diagonal
SPRINGDAMPINGCONSTANT 2.0f      // Dämpfungskonstante
DRAGCOEFFICIENT   0.1f     // Luftwiderstandskoeffizient
WINDFACTOR        5        // Windfaktor
CLOTHMASS         35.0f    // Gesamtmasse der Kleidung
```

Für ein wesentlich kleineres Mesh der Größe

IWIDTH	10
ILENGTH	10
IRESWIDTH	11
IRESLENGTH	11

können auch die anderen Parameter und Konstanten niedriger gewählt werden:

GRAVITY	-9.81	// Einheit: m/s <sup>2</sup>
SPRINGTENSIONCONSTANT	17.0f	// Federkonstante vertikal & horizontal
SPRINGSHEARCONSTANT	20.0f	// Federkonstante diagonal
SPRINGDAMPINGCONSTANT	1.0f	// Dämpfungskonstante
DRAGCOEFFICIENT	0.01f	// Luftwiderstandskoeffizient
WINDFACTOR	3	// Windfaktor
CLOTHMASS	10.0f	// Gesamtmasse der Kleidung

Der größte Unterschied ist in den Federkonstanten zu sehen. Wenn ich die Konstanten des kleinen Meshes auch für das große verwende, dehnen sich die Federn umso mehr, was bedeutet: je größer die Federkonstante, desto steifer das Mesh. Solange die Federn richtig gewählt werden, ist die Simulation robust.

Die Größe des Zeitschritts hängt auch ein wenig von der Größe des Meshes ab: bei dem kleineren habe ich den Zeitschritt auf 10 Millisekunden festgesetzt, bei dem größeren auf 16 Millisekunden.

### 3.3.3. Integration: Eulermethode erster Ordnung

Für die Simulation des Meshes habe ich die Eulermethode erster Ordnung aufgrund ihrer Einfachheit verwendet (siehe oben). Mit Hilfe der Funktion `StepSimulation(float dt)` werden die Bewegungsgleichungen integriert. Zunächst wird die Methode `UpdateMesh()` aufgerufen, in der alle Kräfte bereits berechnet wurden. Daraufhin werden sie integriert, falls das Partikel nicht fixiert ist:

- Aufgrund der Kräfte kann eine Beschleunigung berechnet werden, die das Produkt aus Kraft und inverser Masse ist.
- Mithilfe der Beschleunigung und einer alten Geschwindigkeit aus `UpdateMesh()` ist die neue Geschwindigkeit das Produkt aus der Beschleunigung und dem Zeitschritt, addiert mit der vorigen Geschwindigkeit.

- Die neue Position des Partikels ist schließlich die Geschwindigkeit multipliziert mit dem Zeitschritt und dann addiert mit der alten Position.

```

void ParticleSystem::StepSimulation (float dt)
{
    float dtime = dt;

    /* alle Kräfte berechnen */
    UpdateMesh();

    /* Integration */
    for ( int b=0; b<iResWidth; b++ )
    {
        for ( int l=0; l<iResLength; l++ )
        {
            if ( mMesh[b][l].bFixed == false )
            {
                mMesh[b][l].setAcceleration ( Vector
                    (mMesh[b][l].getForces() /
                    mMesh[b][l].getMass() ));

                mMesh[b][l].setVelocity ( Vector
                    (mMesh[b][l].getVelocity() +
                    mMesh[b][l].getAcceleration() * dtime ));

                mMesh[b][l].setPosition ( Vector
                    (mMesh[b][l].getPosition() +
                    mMesh[b][l].getVelocity() * dtime ));
            }
        }
    }
}

```

Die Simulation wird über die Taste “s” ausgelöst und funktioniert nur solange die Taste gedrückt ist. Lässt man die Taste los, stoppt die Simulation, betätigt man sie erneut, wird die Simulation an der gestoppten Stelle fortgesetzt.

## **4. Ergebnisse**

Mit dieser einen Simulation konnte ich verschiedene Ergebnisse erzielen. Dafür musste ich mir lediglich die Punkte fixieren, die sich nicht bewegen sollten. Die Fixierung kann man beliebig in der Implementierung ändern.

Für die ersten beiden Beispiele, das fallende Tuch und die Tischdecke, werden die Schwerkraft, der Reibungsluftwiderstand und die Feder-Dämpfungskraft benötigt. Bei dem letzten Beispiel, der Fahne, kommt noch zusätzlich die Windkraft mit ins Spiel.

### **4.1. fallendes Tuch**

Für das fallende Tuch habe ich in der Methode `GenerateMesh()` den letzten Punkt des Meshes fixiert:

```
if ((b==iResWidth-1) && (l==iResLength-1))
    mMesh[b][l].bFixed = true;
else
    mMesh[b][l].bFixed = false;
```

Sobald ich die Simulation starte, fällt das Stück Stoff aus der Ruheposition zunächst größtenteils waagrecht nach unten. Doch je weiter es fällt, desto mehr Federn stellen sich senkrecht, da sie ja die Partikel mit dem fixierten Punkt verbinden.

Bei dieser Simulation habe ich unterschieden zwischen einem fallenden Tuch ohne Luftwiderstand und einem fallenden Tuch mit Luftwiderstand. Solange das Tuch fällt macht sich kaum ein Unterschied bemerkbar, ganz leicht jedoch an den Kanten.

Allerdings ziehen bei der Simulation ohne Luftwiderstand die Federn das Tuch wieder ganz nach oben, sogar über den Punkt hinaus, der fixiert ist. Danach fällt es wieder hinunter. Das fallende Tuch mit Luftwiderstand hängt sich viel schneller aus, kommt also viel schneller in den Ruhezustand. Mit Hilfe der Abbildungen 35 bis 58 lässt sich das Ganze leichter verstehen.

### **4.2. Tischdecke**

Das Stück Stoff, das als Tischdecke dienen soll, ist ähnlich konzipiert wie das fallende Tuch. Die Ruheposition ist genau die Gleiche, aber für die Tischdecke habe

ich nicht den letzten Punkt fixiert, sondern eine quadratische Fläche – sozusagen ein imaginärer Tisch – in der Mitte des Meshes:

```
if ( (b >= IRESWIDTH/4) && (b <= IRESWIDTH*3/4)
    && (l >= IRESLENGTH/4) && (l <= IRESLENGTH*3/4) )
{
    mMesh[b][l].bFixed = true;
}
else
    mMesh[b][l].bFixed = false;
```

Wenn die Taste “s” betätigt wird, fallen alle Punkte um die fixierte Fläche herum, ähnlich wie bei dem vorigen Beispiel, nach unten. Hierbei fällt allerdings der Stoff an einer Seite gleichzeitig zu den drei anderen Seite hinunter. Je weiter außen die Partikel liegen, desto länger bleiben sie in der Horizontalen.

Auch hier hab ich wieder die Tischdecke mit und ohne Luftwiderstand verglichen. Lasse ich den Luftwiderstand weg, so schwingt das Stück Stoff viel weiter nach oben hin zurück, ein wenig über die fixierte Fläche hinaus. Diese Simulation wirkt so, als würde von unten der Wind den Stoff nach oben blasen. Die Tischdecke mit Luftwiderstand sieht dagegen etwas realistischer aus. Die Abbildungen 59 bis 86 belegen dies eindeutig.

### **4.3. Fahne**

Um nicht die Position des gesamten Meshes ändern zu müssen, habe ich nicht eine Fahne, die vertikal aufgehängt ist, sondern horizontal (Abb. 87 bis 102). Ich habe wieder einige Punkte fixiert, in diesem Fall äußere Reihe von Partikeln, die an einer imaginären Fahnenstange befestigt sind:

```
if ((l == 0)) {
    mMesh[b][l].bFixed = true;
}
else
    mMesh[b][l].bFixed = false;
```

Der Wind kommt von rechts und leicht von oben. Wie oben bereits genauer erläutert, wird eine gewisse Zufälligkeit in die Windstärke mit einberechnet. Mit Hilfe der Abbildungen sieht man, dass dadurch die Fahne nicht gleichmäßig weht, sondern der Wind immer mal wieder kurz nachlässt und wieder stärker wird, wodurch das ganze realistischer wirkt.

## **5. Fazit und Ausblick**

Ziel meiner Arbeit war es, ein Stück Stoff möglichst real auf dem Bildschirm darzustellen. Ich musste mich dazu entweder für ein Kontinuum-Mechanik-Modell oder ein Partikelsystem-Modell entscheiden. Meine Wahl fiel auf ein Partikelsystem, denn es hat den Vorteil, dass es qualitativ richtige Ergebnisse liefert. Noch dazu ist eine schnellere Berechnung möglich und es ist einfacher zu implementieren.

Für die Simulation habe ich aufgrund ihrer Einfachheit die Euler-Methode verwendet. Sie ist besonders gut geeignet für einfache Partikelsysteme mit regulären Meshes und einfachen, linearen Interaktionen zwischen den Partikeln. Somit lässt sich die Bewegung jedes Partikels genau berechnen.

Nachdem das System so gut wie fertig war, musste ich noch die Konstanten richtig wählen. Die Stabilität des Systems ist von mehreren Konstanten gleichzeitig abhängig: der Zeit, den Federkonstanten, den Parametern der verschiedenen Kräfte. Doch nach einigen Tests war auch dieses Problem gelöst. Damit sie jederzeit schnell veränderbar sind, habe ich sie global definiert.

Mit meinem Partikelsystem lässt sich momentan ein fallendes Tuch, das an beliebiger Stelle aufgehängt werden kann, realisieren. Außerdem habe ich noch eine Tischdecke und eine Fahne implementiert, die sich auch mit einer höheren Auflösung des Meshes realisieren lassen. Die Stoffart bzw. die Stoffmasse ist bei allen drei Beispielen identisch, genauso die Farbe. In dieser Hinsicht wäre mein System noch erweiterbar: man könnte verschiedene Stoffarten, sowie unterschiedliche Texturen verwenden. Durch einige Ergänzungen und zum Teil Veränderungen wäre mit diesem System auch die Bekleidung eines virtuellen Menschen denkbar.

## 6. Abbildungen

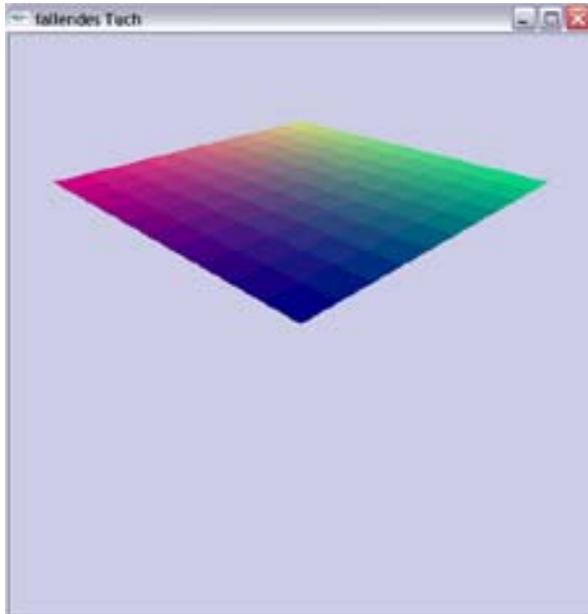


Abb.35:  
fallendes Tuch  
Nr.1

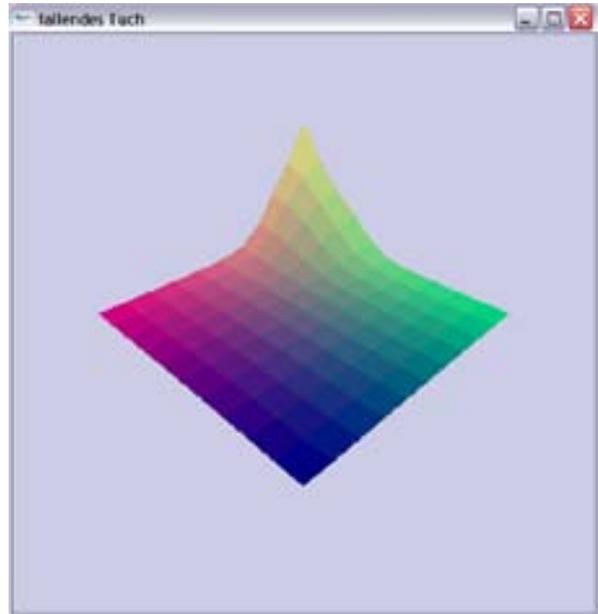


Abb.36:  
fallendes Tuch  
Nr.2

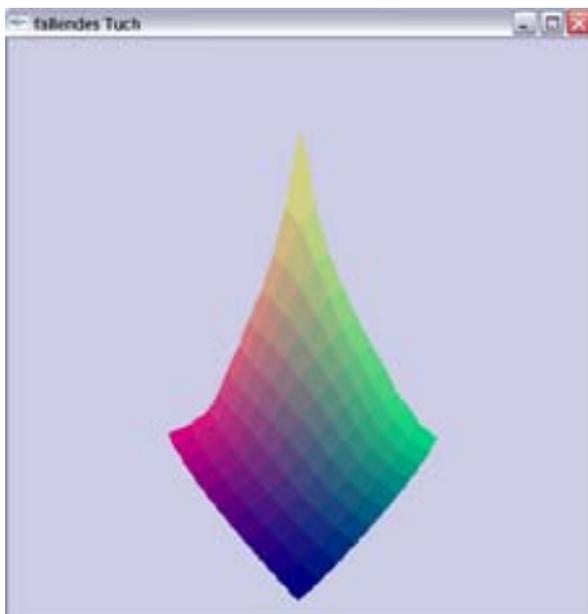


Abb.37:  
fallendes Tuch  
Nr.3

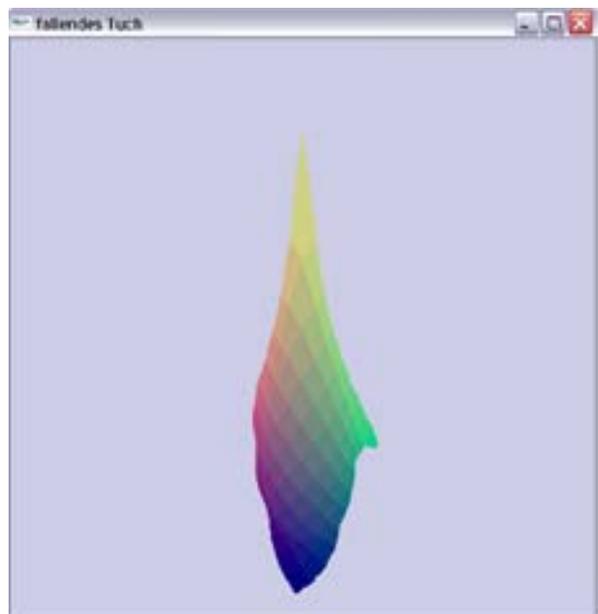


Abb.38:  
fallendes Tuch  
Nr.4

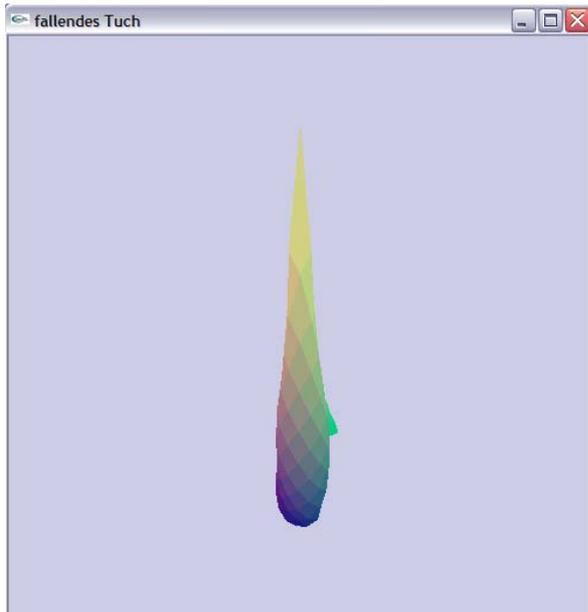


Abb.39:  
fallendes Tuch  
Nr.5

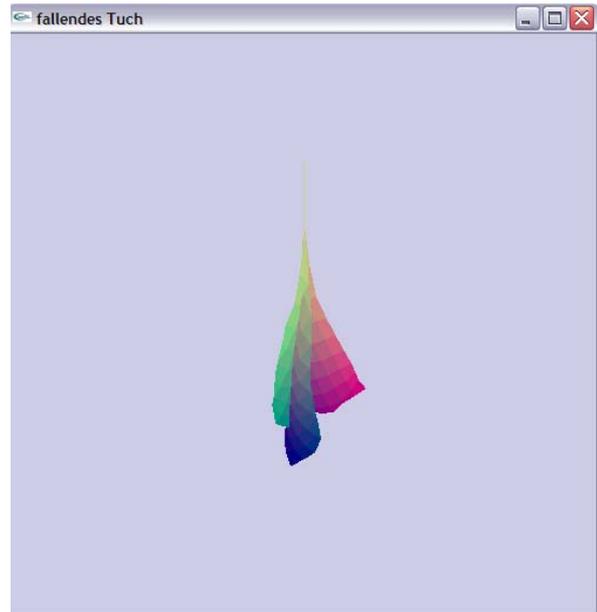


Abb.40:  
fallendes Tuch  
Nr.6

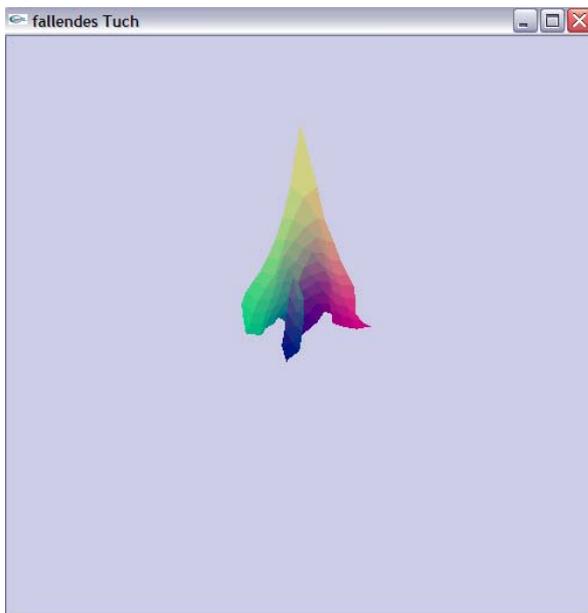


Abb.41:  
fallendes Tuch  
Nr.7

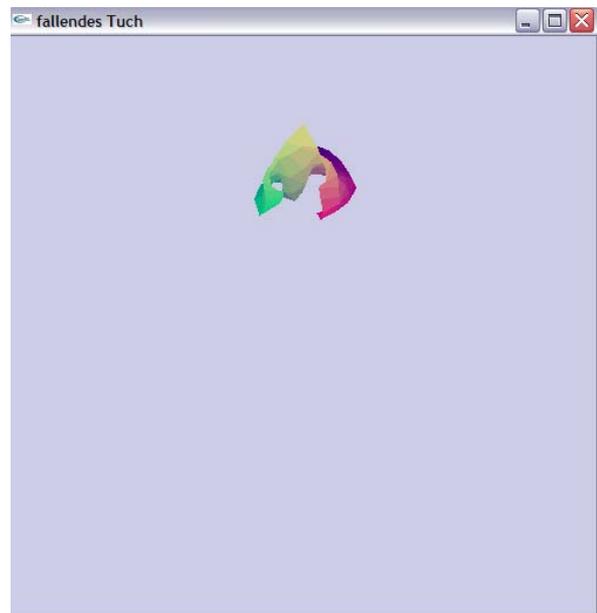


Abb.42:  
fallendes Tuch  
Nr.8

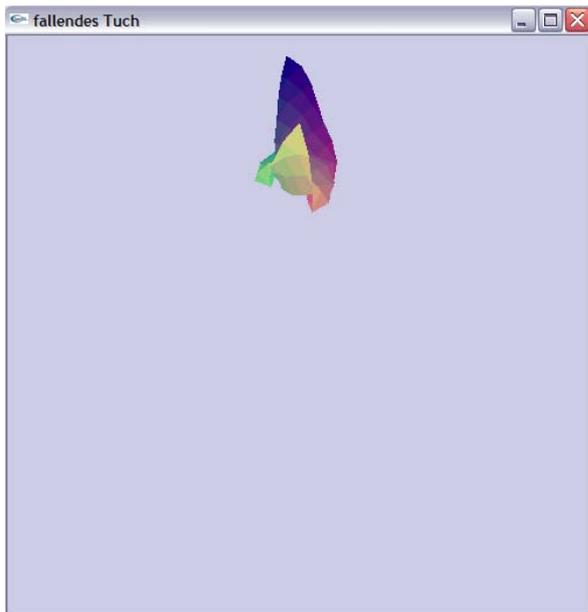


Abb.43:  
fallendes Tuch  
Nr.9

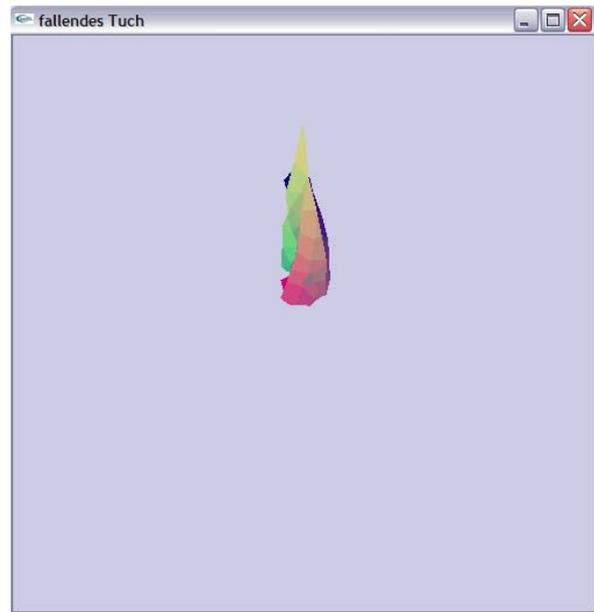


Abb.44:  
fallendes Tuch  
Nr.10

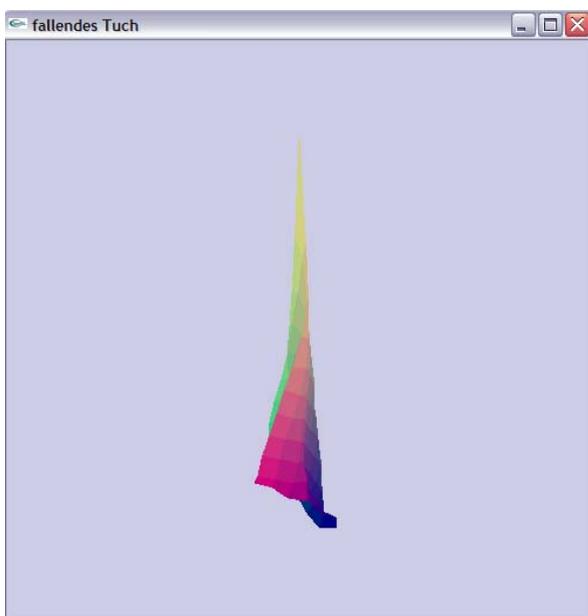


Abb.45:  
fallendes Tuch  
Nr.11

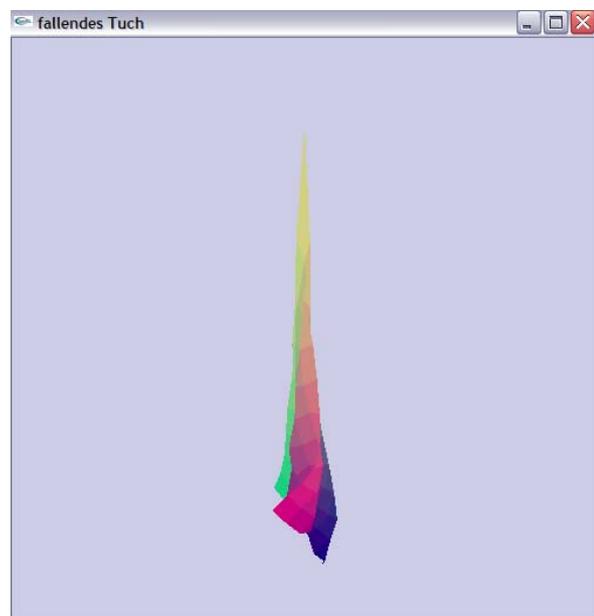


Abb.46:  
fallendes Tuch  
Nr.12

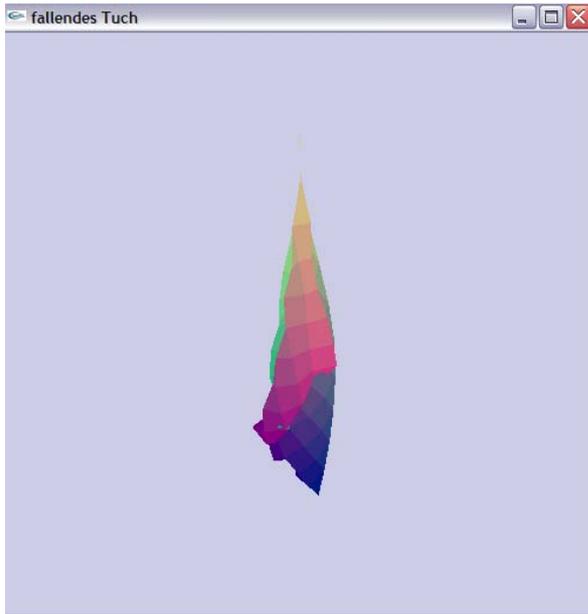


Abb.47:  
fallendes Tuch  
Nr.13

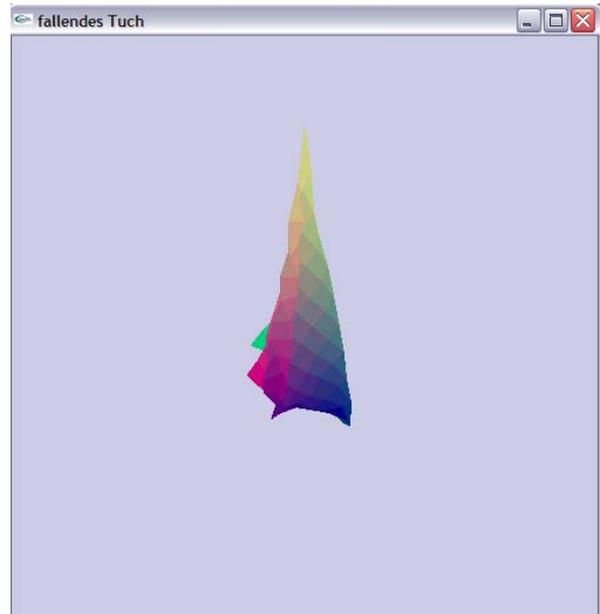


Abb.48:  
fallendes Tuch  
Nr.14

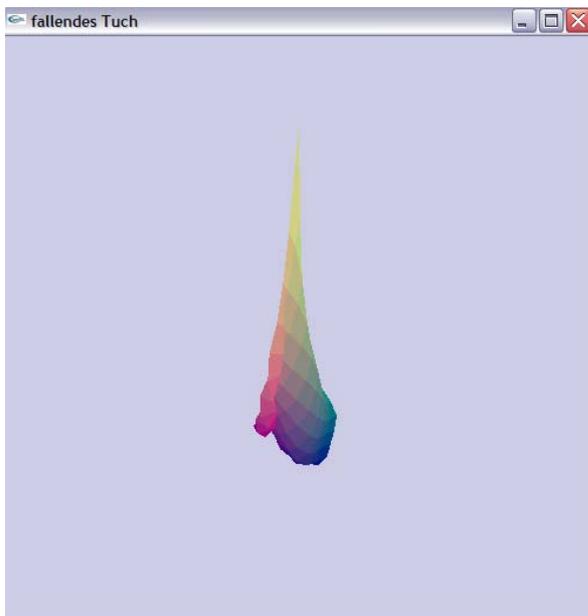


Abb.49:  
fallendes Tuch  
Nr.15

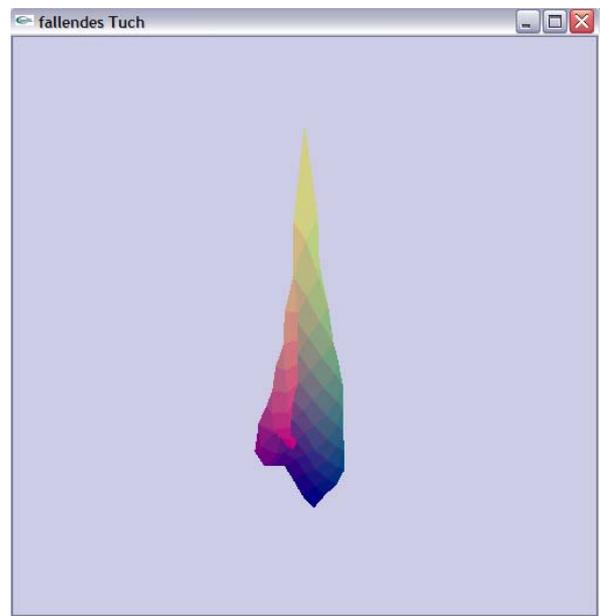


Abb.50:  
fallendes Tuch  
Nr.16

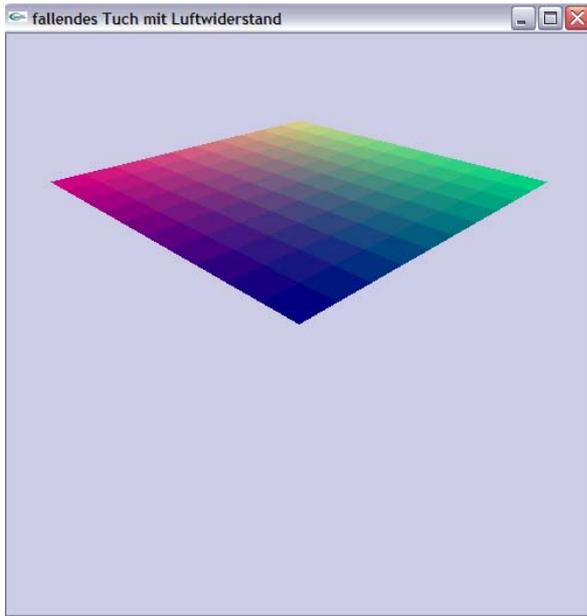


Abb.51:  
fallendes Tuch  
mit Luftwiderstand  
Nr.1

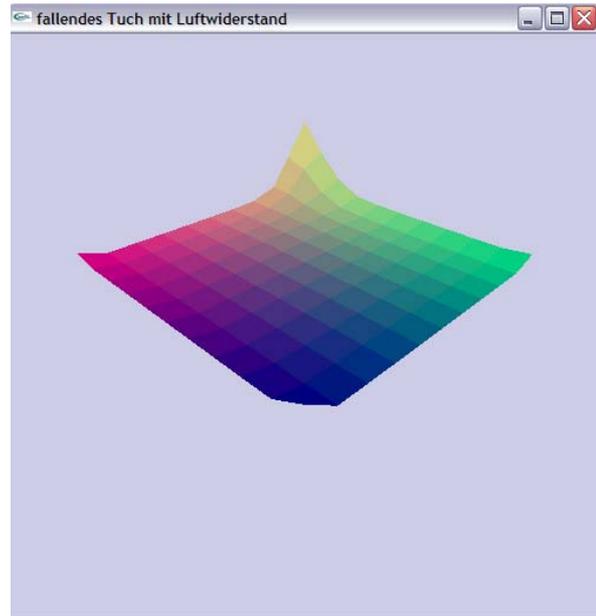


Abb.52:  
fallendes Tuch  
mit Luftwiderstand  
Nr.2

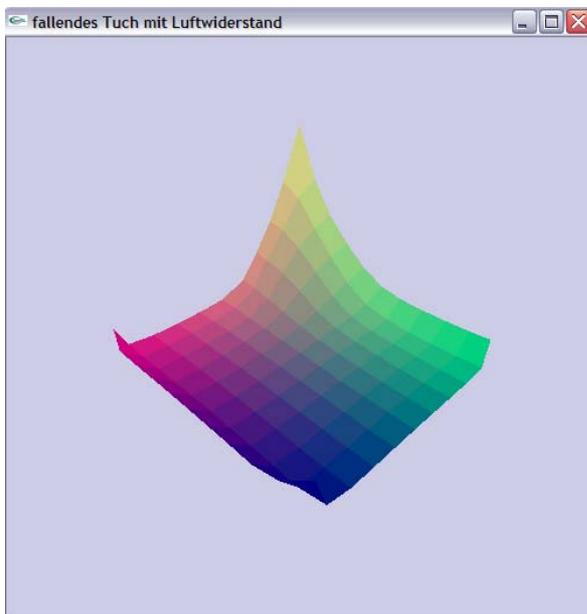


Abb.53:  
fallendes Tuch  
mit Luftwiderstand  
Nr.3

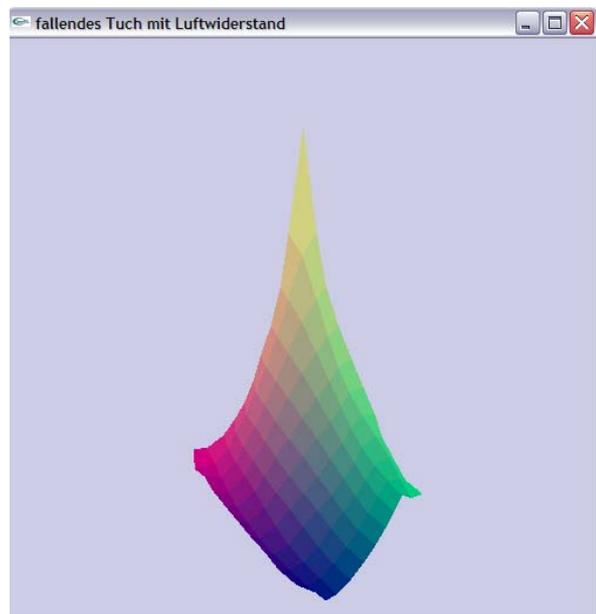


Abb.54:  
fallendes Tuch  
mit Luftwiderstand  
Nr.4

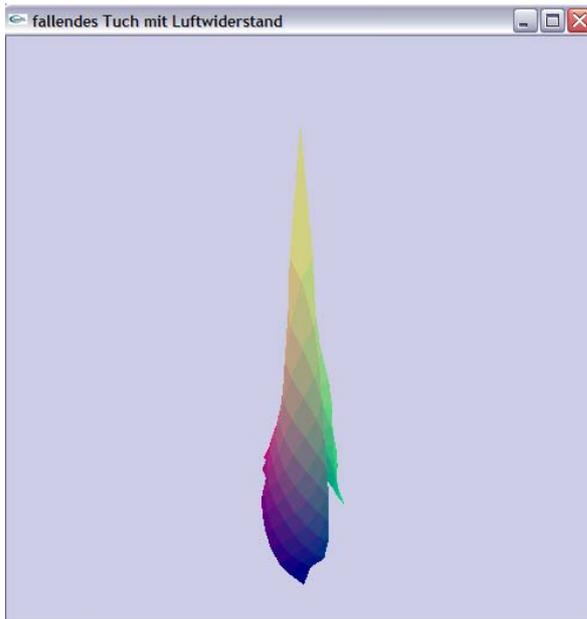


Abb.55:  
fallendes Tuch  
mit Luftwiderstand  
Nr.5

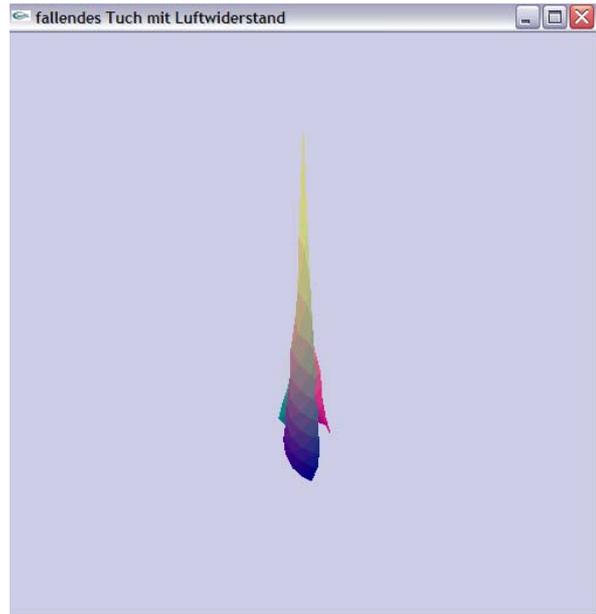


Abb.56:  
fallendes Tuch  
mit Luftwiderstand  
Nr.6

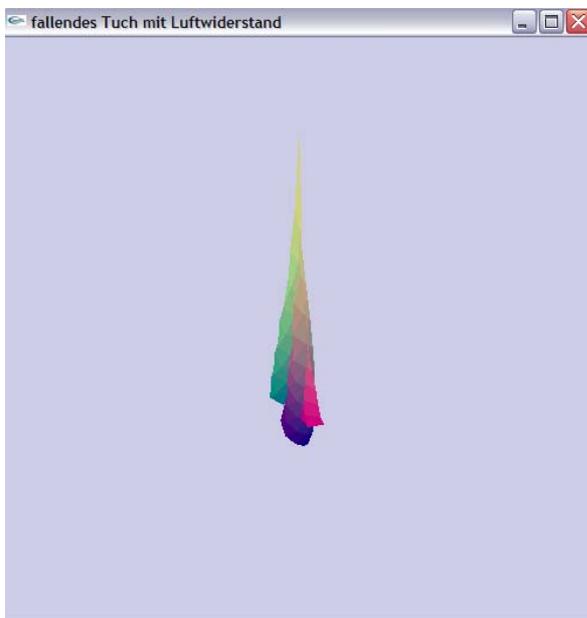


Abb.57:  
fallendes Tuch  
mit Luftwiderstand  
Nr.7

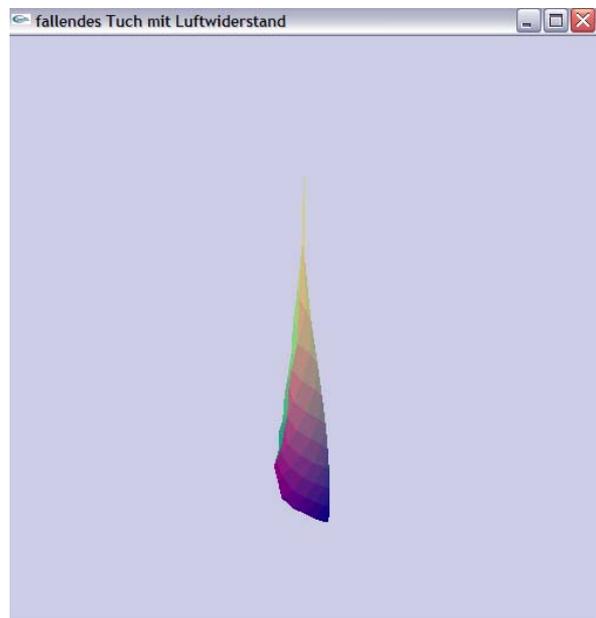


Abb.58:  
fallendes Tuch  
mit Luftwiderstand  
Nr.8

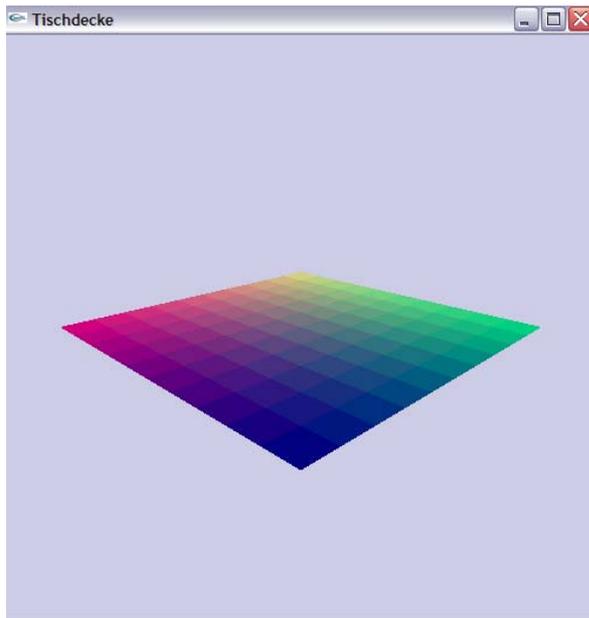


Abb.59:  
Tischdecke  
Nr.1

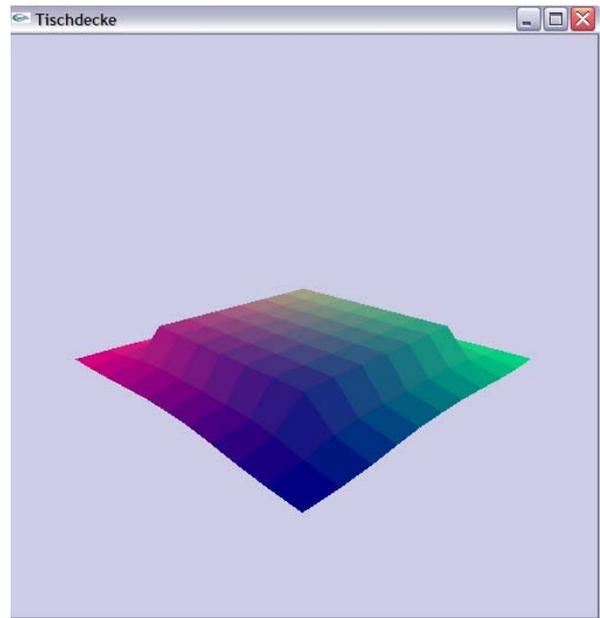


Abb.60:  
Tischdecke  
Nr.2

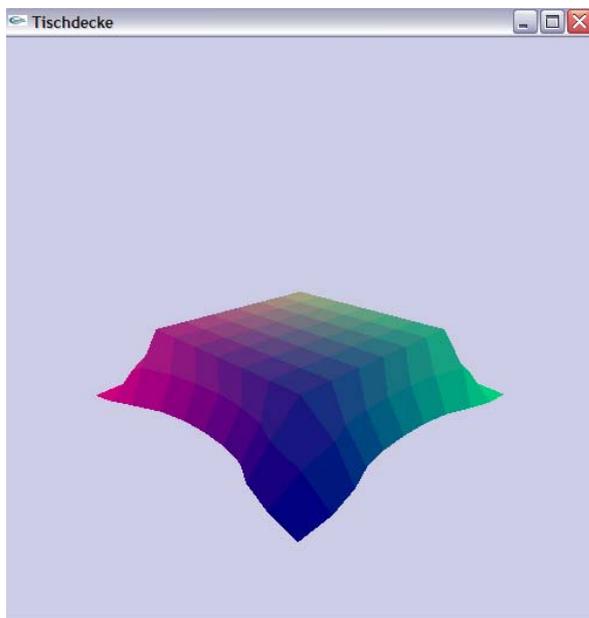


Abb.61:  
Tischdecke  
Nr.3

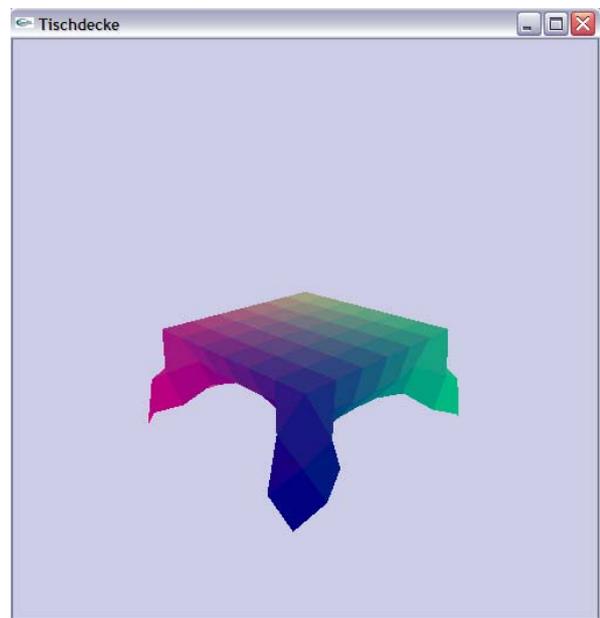


Abb.62:  
Tischdecke  
Nr.4

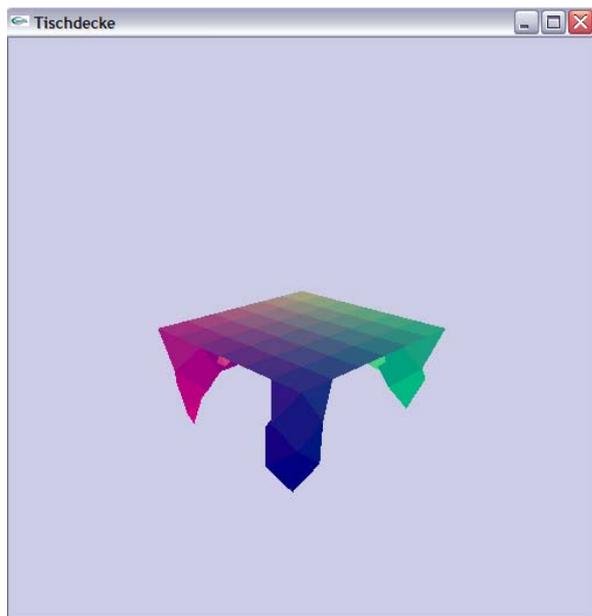


Abb.63:  
Tischdecke  
Nr.5

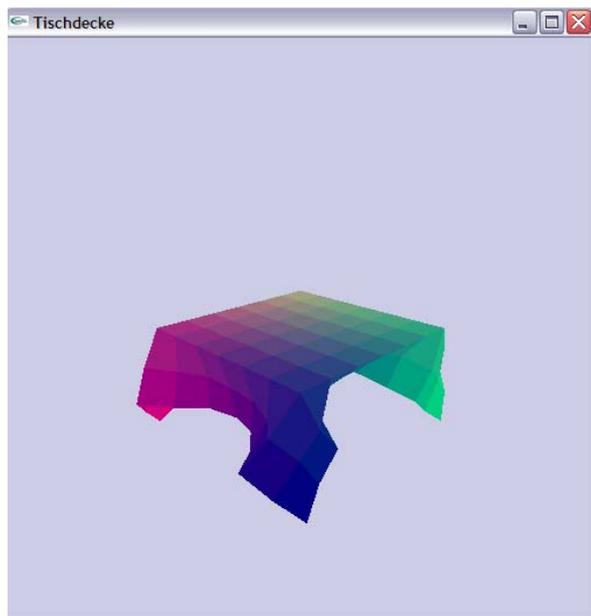


Abb.64:  
Tischdecke  
Nr.6

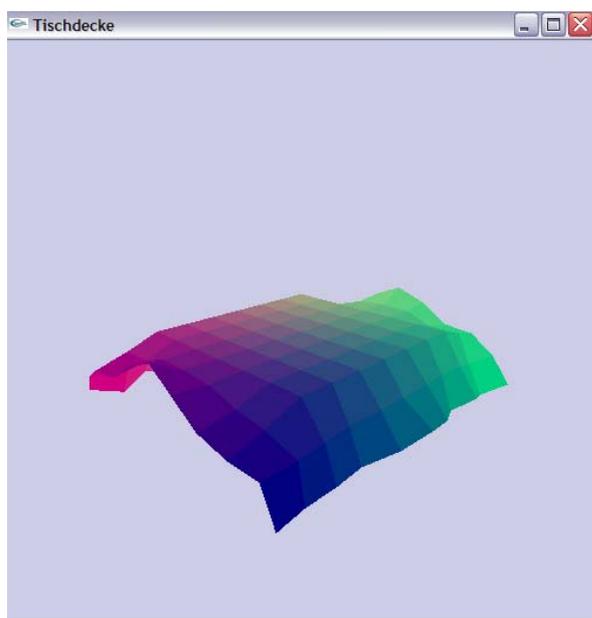


Abb.65:  
Tischdecke  
Nr.7

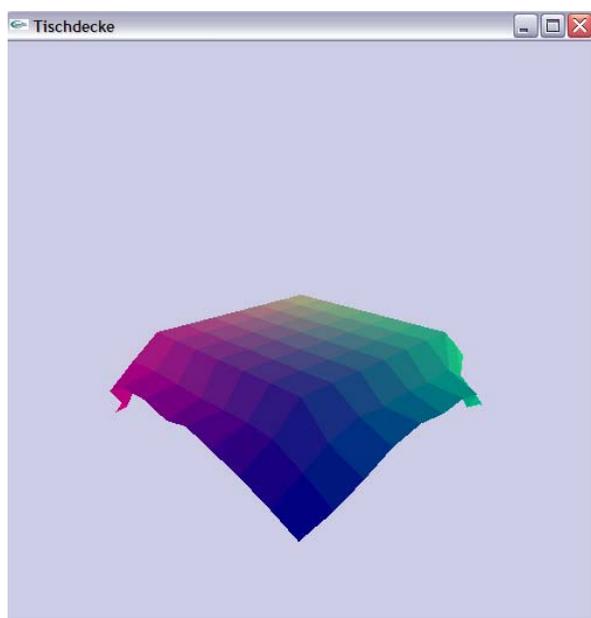


Abb.66:  
Tischdecke  
Nr.8

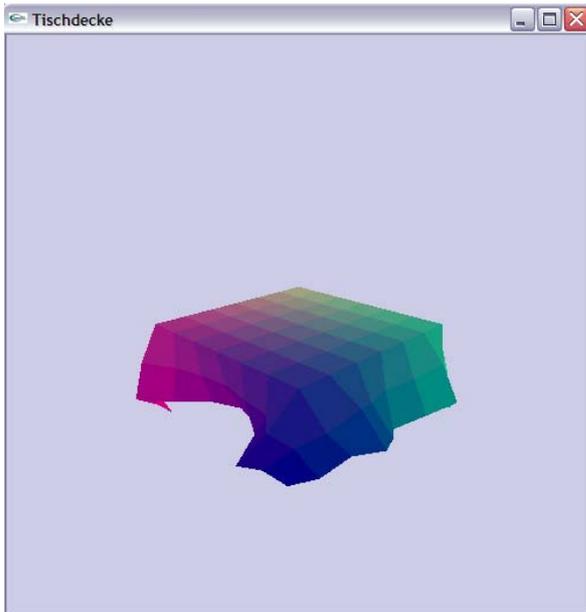


Abb.67:  
Tischdecke  
Nr.9

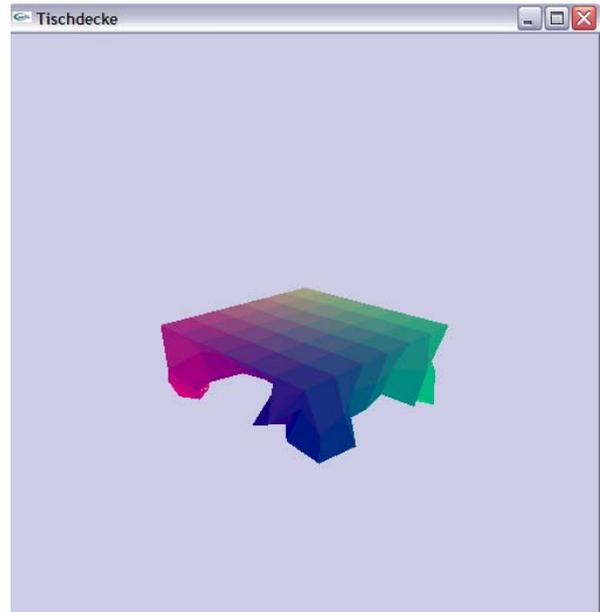


Abb.68:  
Tischdecke  
Nr.10

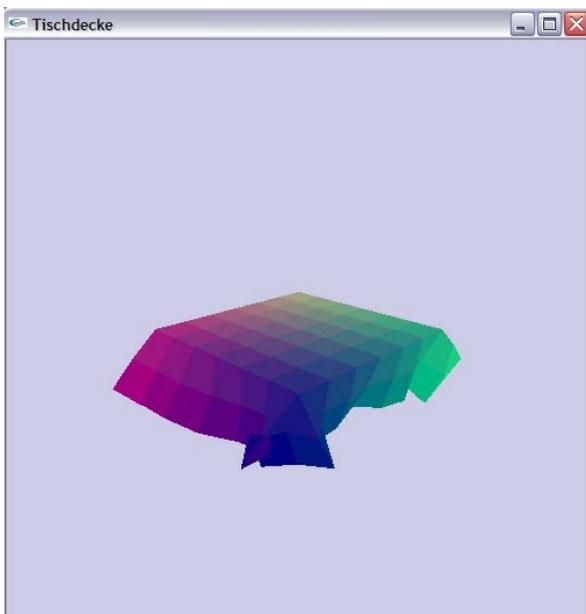


Abb.69:  
Tischdecke  
Nr.11

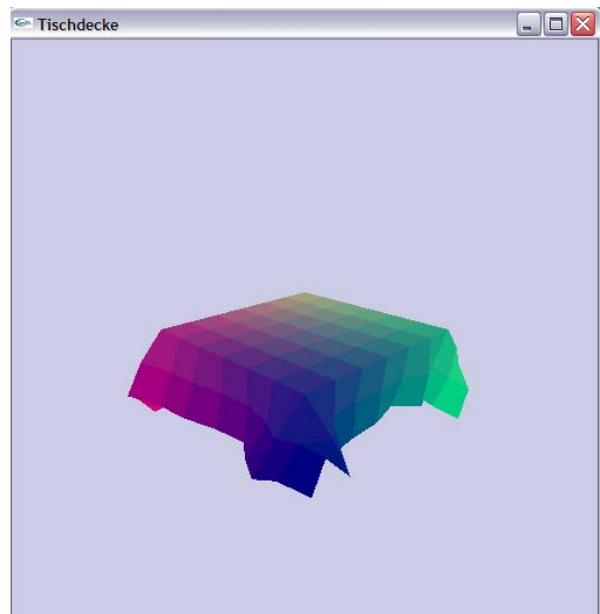


Abb.70:  
Tischdecke  
Nr.12

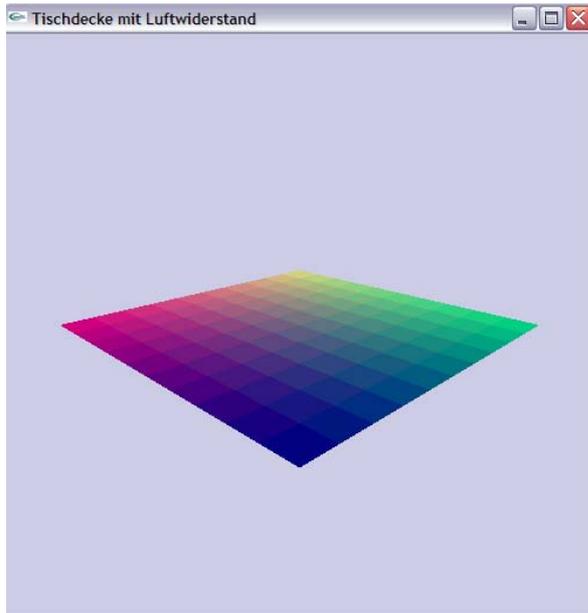


Abb.71:  
Tischdecke  
mit Luftwiderstand  
Nr.1

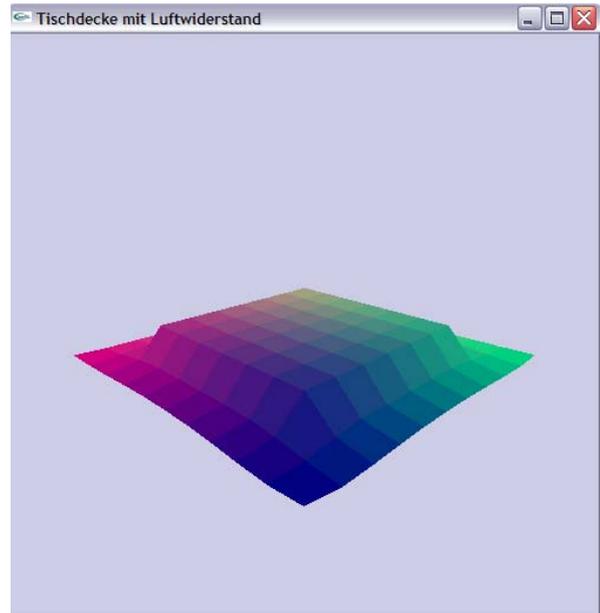


Abb.72:  
Tischdecke  
mit Luftwiderstand  
Nr.2

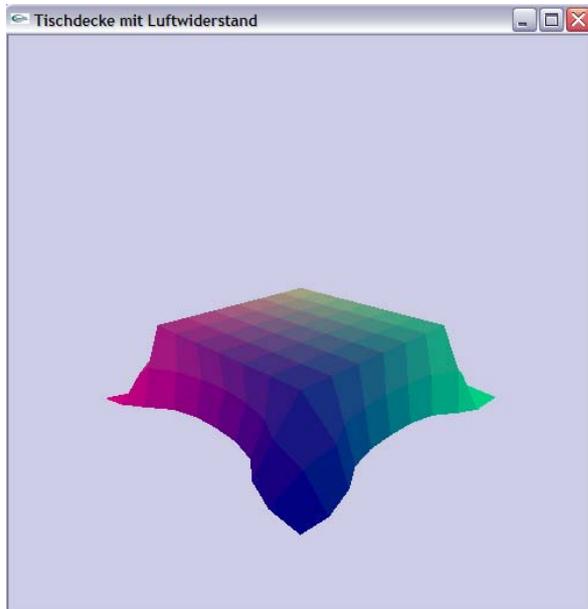


Abb.73:  
Tischdecke  
mit Luftwiderstand  
Nr.3



Abb.74:  
Tischdecke  
mit Luftwiderstand  
Nr.4

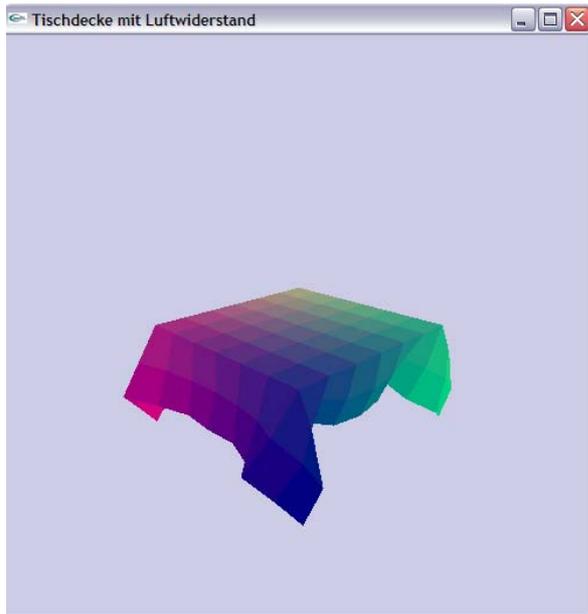


Abb.75:  
Tischdecke  
mit Luftwiderstand  
Nr.5

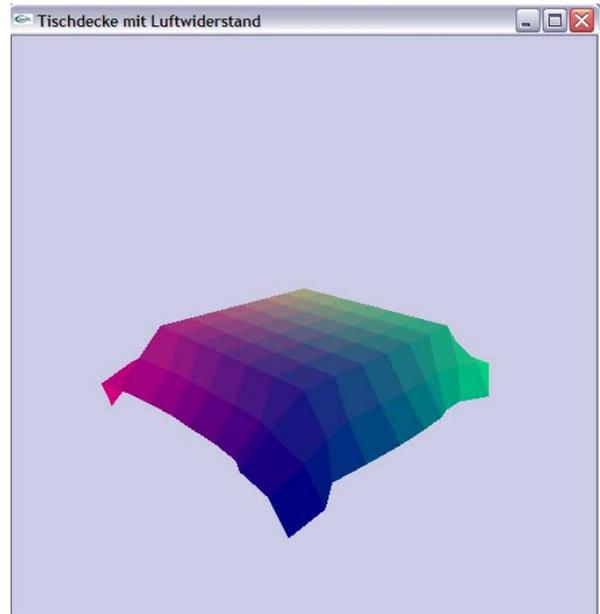


Abb.76:  
Tischdecke  
mit Luftwiderstand  
Nr.6

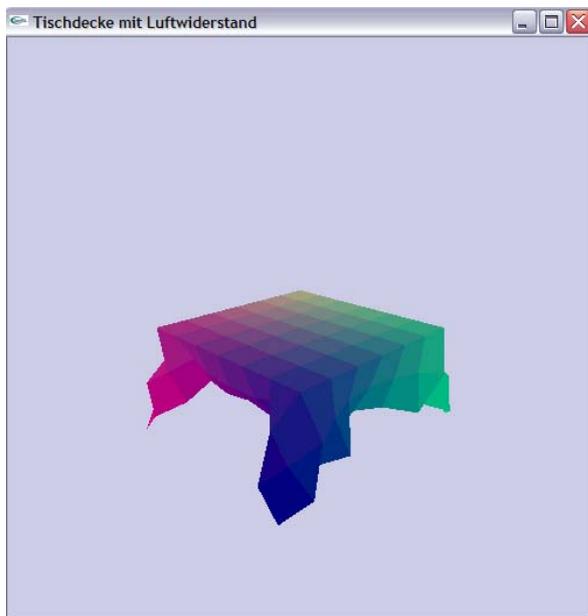


Abb.77:  
Tischdecke  
mit Luftwiderstand  
Nr.7



Abb.78:  
Tischdecke  
mit Luftwiderstand  
Nr.8

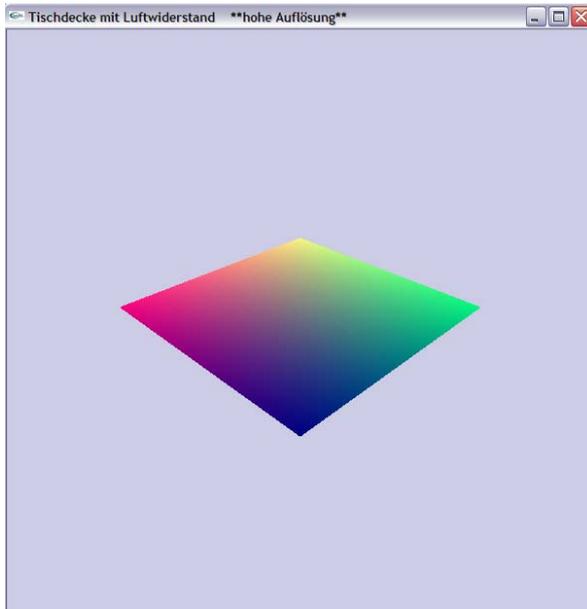


Abb.79:  
Tischdecke  
mit Luftwiderstand  
Nr.1  
(hohe Auflösung)

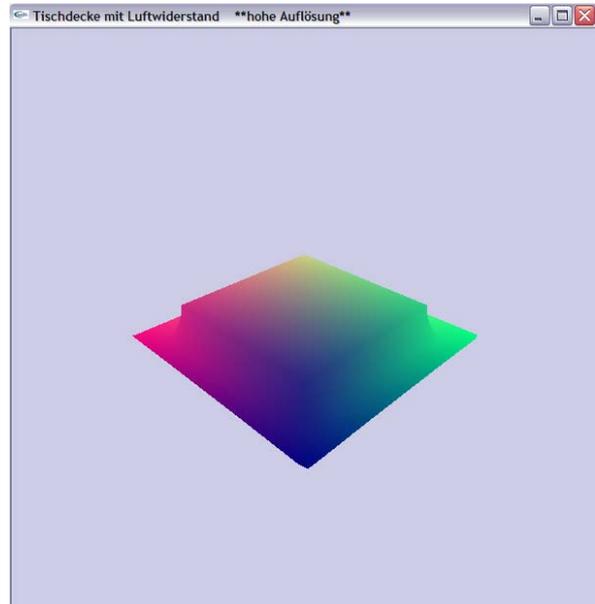


Abb.80:  
Tischdecke  
mit Luftwiderstand  
Nr.2  
(hohe Auflösung)

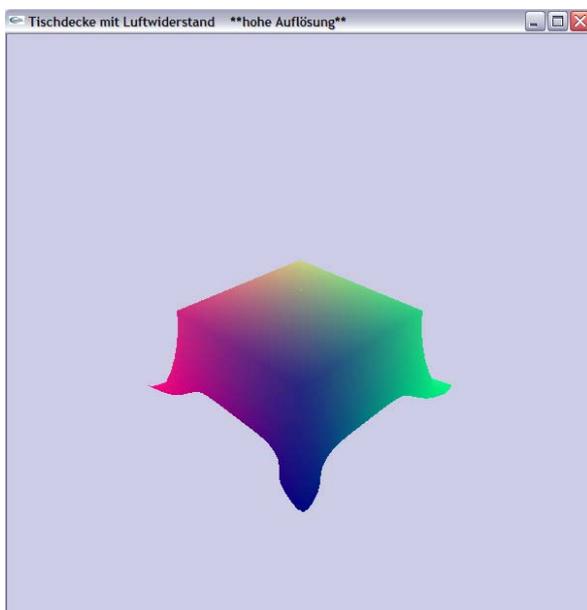


Abb.81:  
Tischdecke  
mit Luftwiderstand  
Nr.3  
(hohe Auflösung)

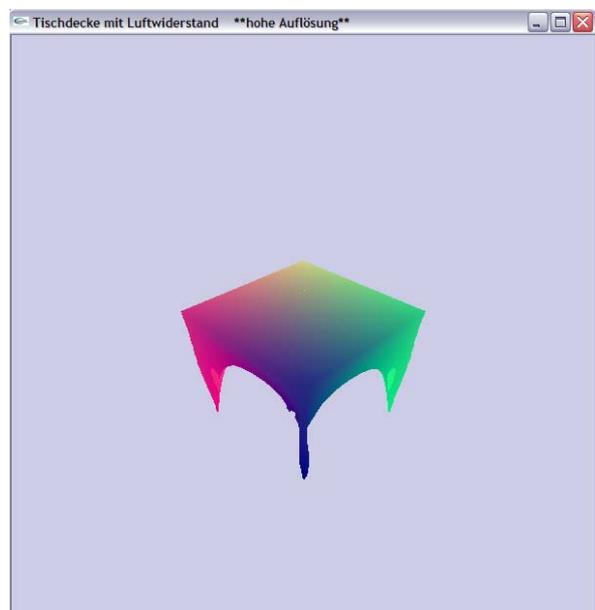


Abb.82:  
Tischdecke  
mit Luftwiderstand  
Nr.4  
(hohe Auflösung)

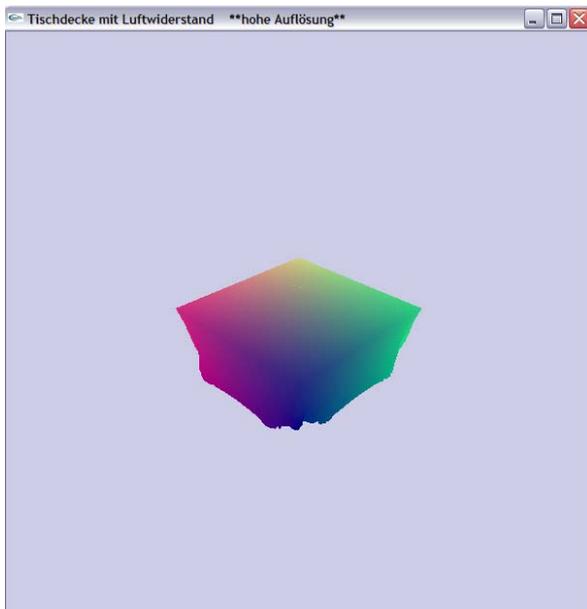


Abb.83:  
Tischdecke  
mit Luftwiderstand  
Nr.5  
(hohe Auflösung)

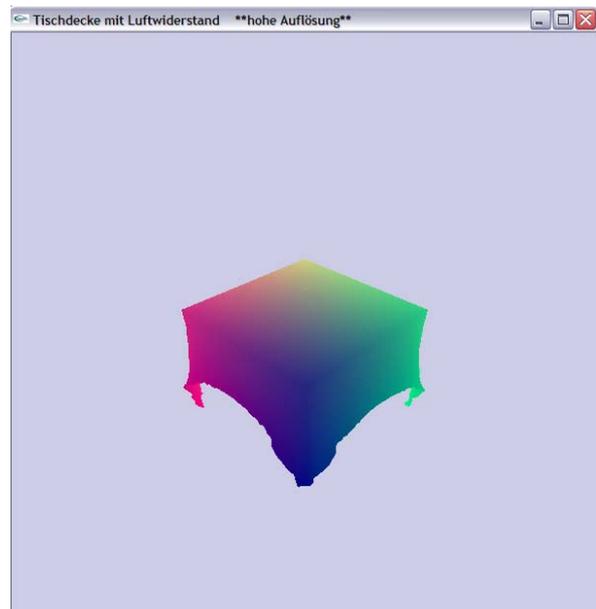


Abb.84:  
Tischdecke  
mit Luftwiderstand  
Nr.6  
(hohe Auflösung)

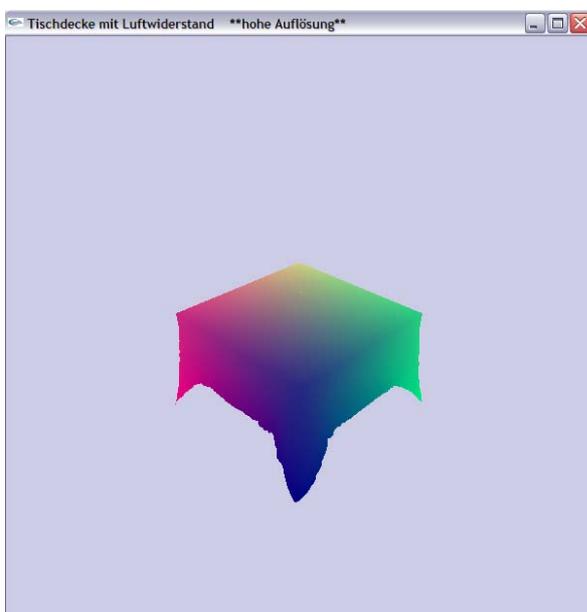


Abb.85:  
Tischdecke  
mit Luftwiderstand  
Nr.7  
(hohe Auflösung)

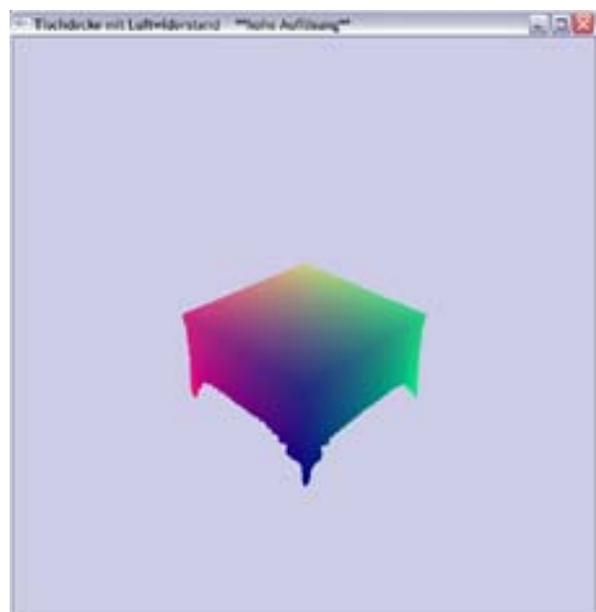


Abb.86:  
Tischdecke  
mit Luftwiderstand  
Nr.8  
(hohe Auflösung)

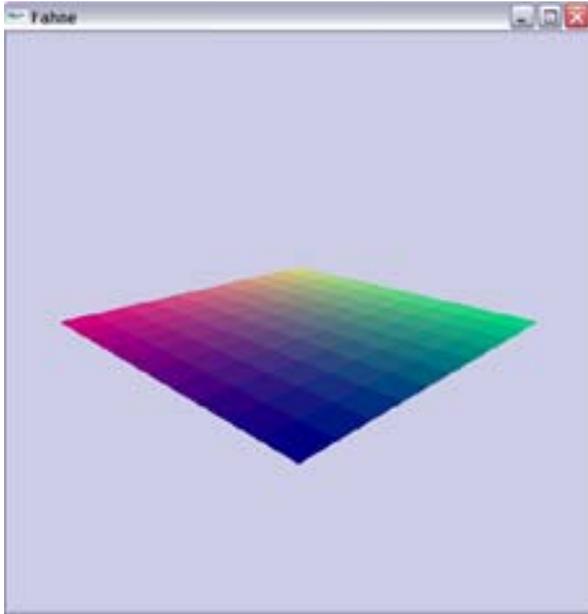


Abb.87:  
Fahne  
Nr.1

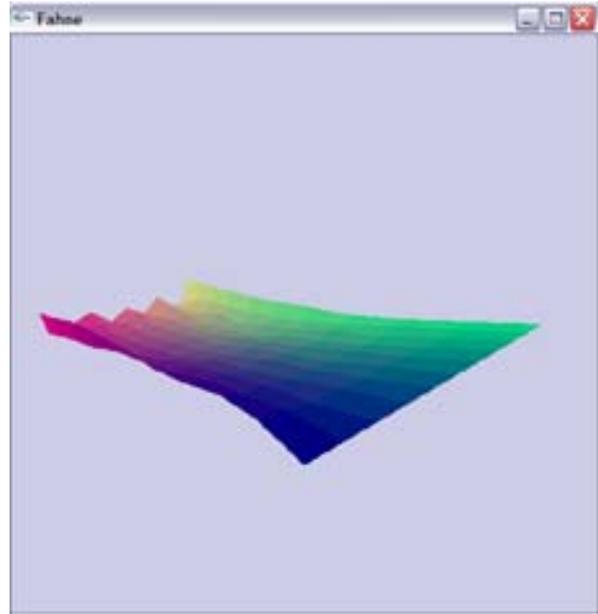


Abb.88:  
Fahne  
Nr.2

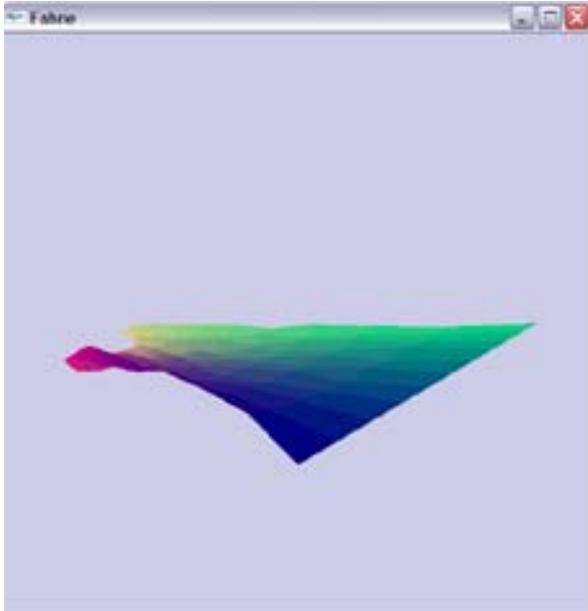


Abb.89:  
Fahne  
Nr.3

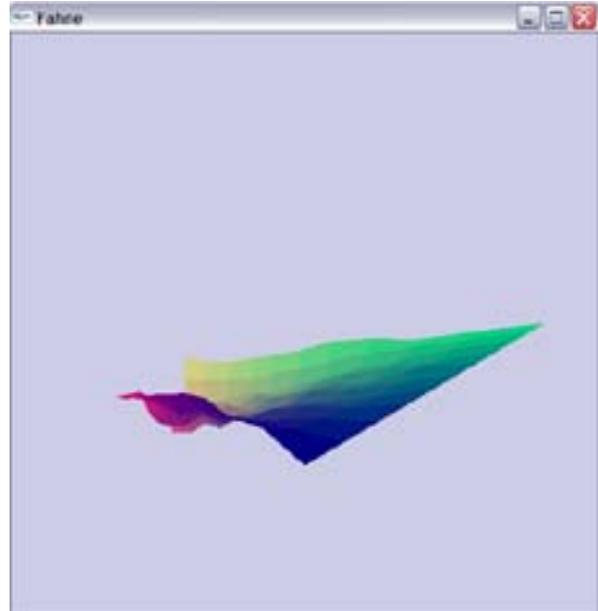


Abb.90:  
Fahne  
Nr.4

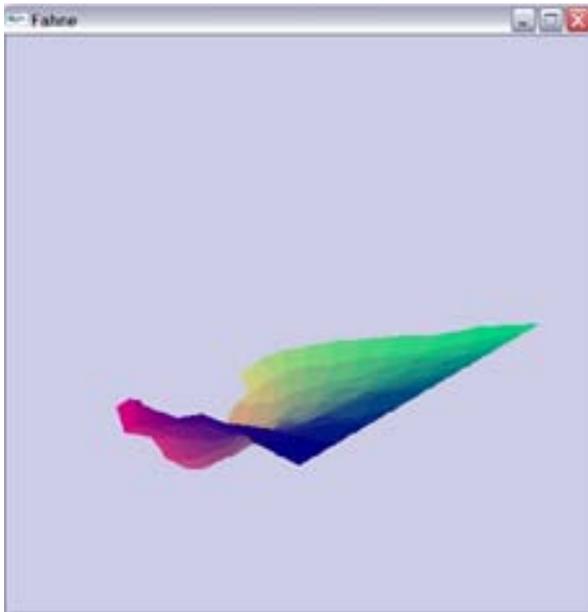


Abb.91:  
Fahne  
Nr.5

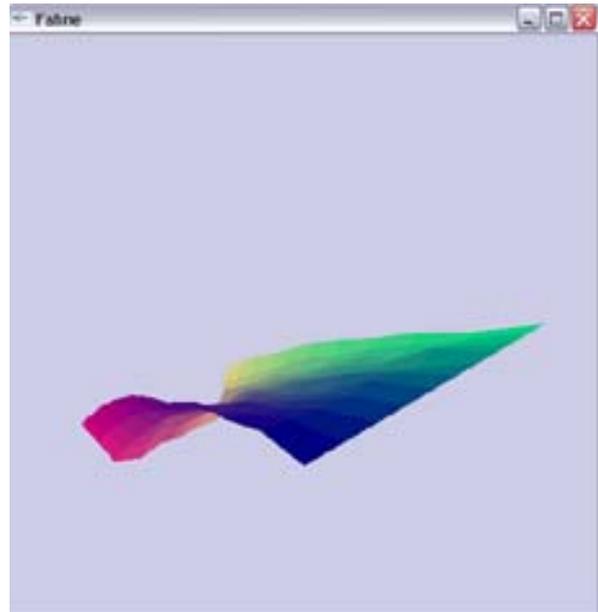


Abb.92:  
Fahne  
Nr.6

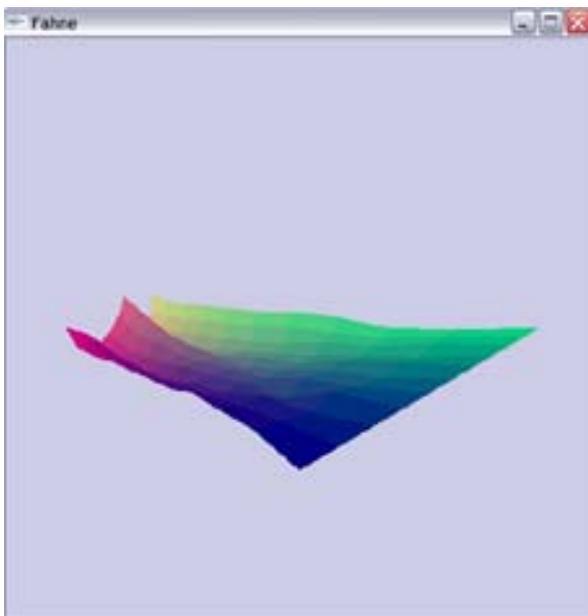


Abb.93:  
Fahne  
Nr.7

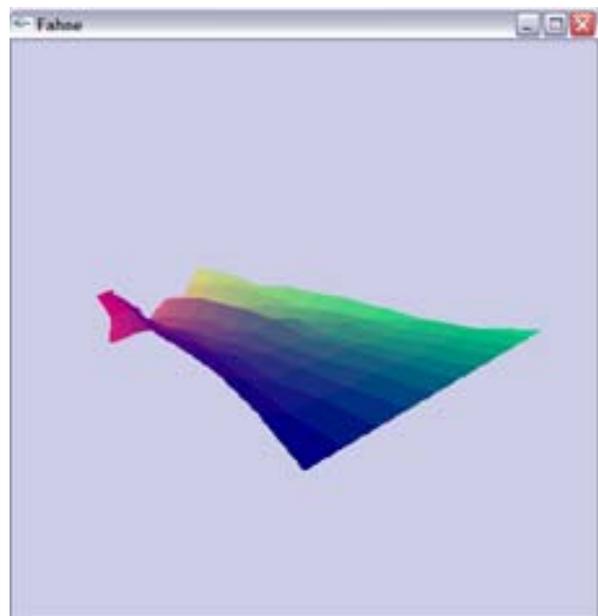


Abb.94:  
Fahne  
Nr.8

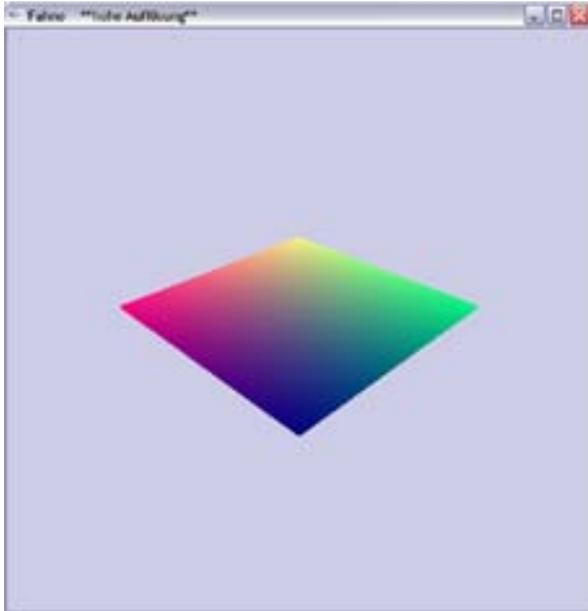


Abb.95:  
Fahne  
Nr.1  
(hohe Auflösung)

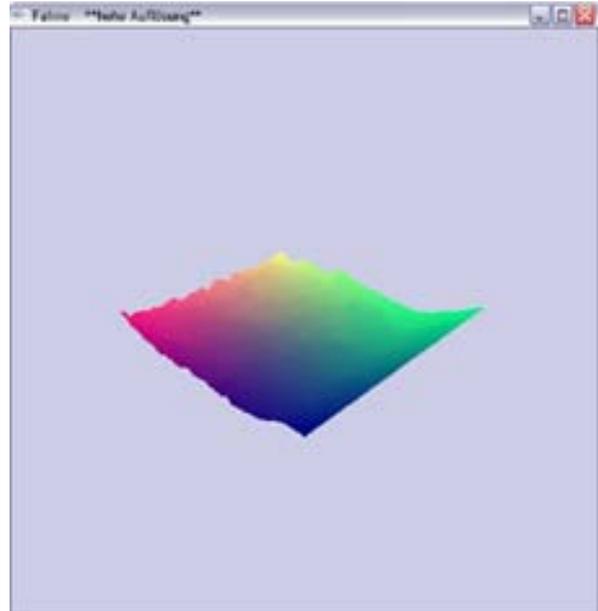


Abb.96:  
Fahne  
Nr.2  
(hohe Auflösung)

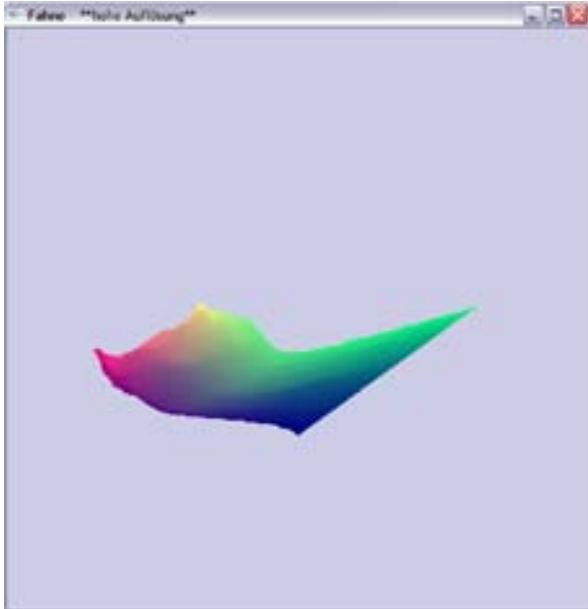


Abb.97:  
Fahne  
Nr.3  
(hohe Auflösung)

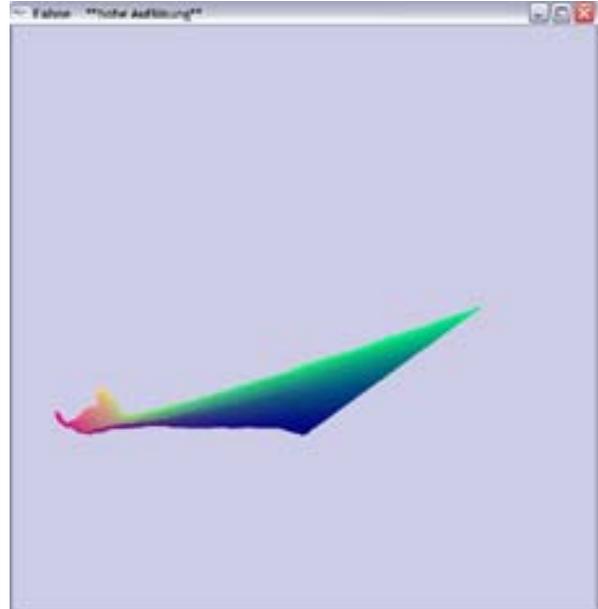


Abb.98:  
Fahne  
Nr.4  
(hohe Auflösung)

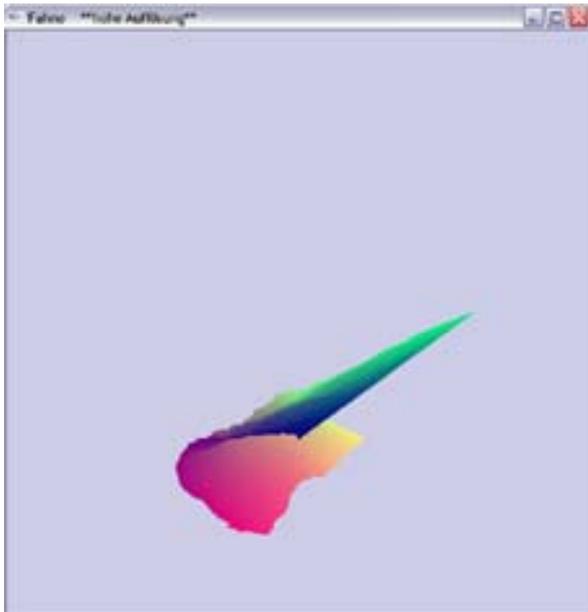


Abb.99:  
Fahne  
Nr.5  
(hohe Auflösung)

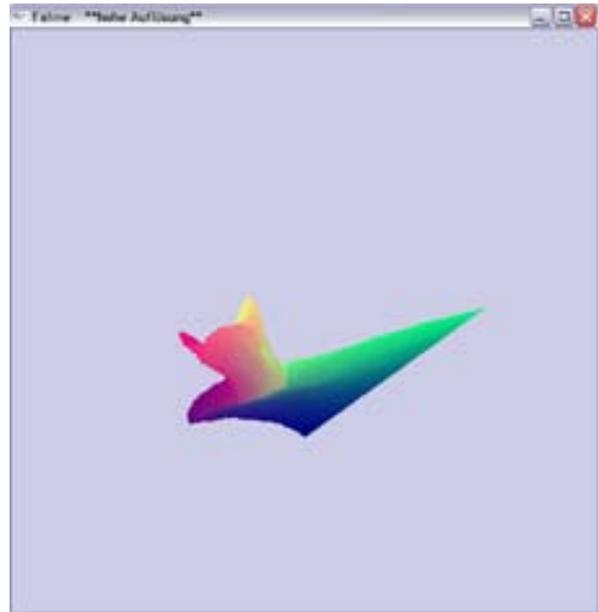


Abb.100:  
Fahne  
Nr.6  
(hohe Auflösung)

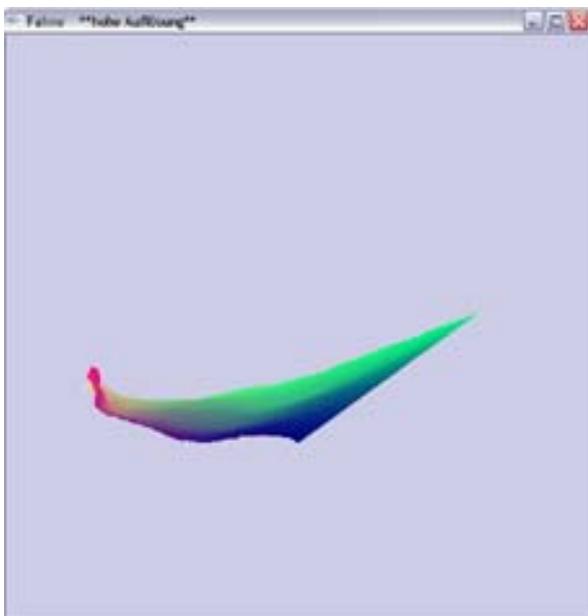


Abb.101:  
Fahne  
Nr.7  
(hohe Auflösung)

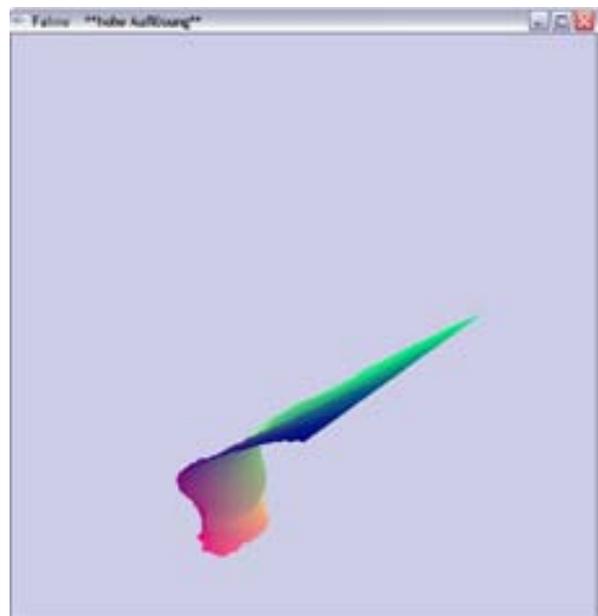


Abb.102:  
Fahne  
Nr.8  
(hohe Auflösung)

## **7. Literaturverzeichnis**

- P. Volino, N. Magnenat-Thalmann, *Virtual Clothing, Theory and Practice*, Springer-Verlag, 2000
- D.H. House, D.E. Breen, *Cloth Modeling and Animation*, A K Peters Ltd., 2000
- D. Bourg, *Physics for Game Developers*, O'Reilly, 2001
- A. Lahoti, *Simulation of Highly Deformable Objects: Cloth Animation*, Indian Institute of Technology Kanpur, April 2000
- *Tutorial T3: Cloth Animation and Rendering*, Eurographics 2002
- D. Baraff, A. Witkin, *Large Steps in Cloth Simulation*, SIGGRAPH 1998
- A. Witkin, *Physically Based Modelling: Principles and Practice*, SIGGRAPH 1997
- L. Chittaro, D. Corvaglia, *3D Virtual Clothing: From Garment Design to Web3D Visualization and Simulation*, HCI Lab, University of Udine